**Canny Edge Detection**

Roshani Narasimhan-201301104

Sai Sruthi Talluri-201301143

**Section 1:**

The canny edge detection is a process which uses multiple steps to detect the edges of an image .The multiple steps are:

1. Smoothen with gaussian filter.
2. Compute horizontal and vertical gradients.
3. Compute magnitude of gradient.
4. Perform Non-Maximal Suppression.
5. Perform Hysteresis Threshold.

**Smoothen with Gaussian Filter:**

Since all edge detection results are easily affected by image noise we convolve the input image with Gaussian kernel(filter) in order to remove high-freq noises. It is useful to remove small-scale textures effectively.

This step will slightly smooth the image to reduce the effects of obvious noise on the edge detector. The equation for a Gaussian filter kernel of size (2*k*+1)×(2*k*+1) is given by:

H_{ij}= \frac{1}{2\pi\sigma^2}\exp(-\frac{(i-k-1)^2+(j-k-1)^2}{2\sigma^2})

We determine size of kernel (odd #) by the value of sigma as follows:

0.0 <= sigma < 0.5 : 3

0.5 <= sigma < 1.0 : 5

1.0 <= sigma < 1.5 : 7

1.5 <= sigma < 2.0 : 9

2.0 <= sigma < 2.5 : 11

2.5 <= sigma < 3.0 : 13 …

kernel Size = 2 \* int(2\*sigma) + 3.

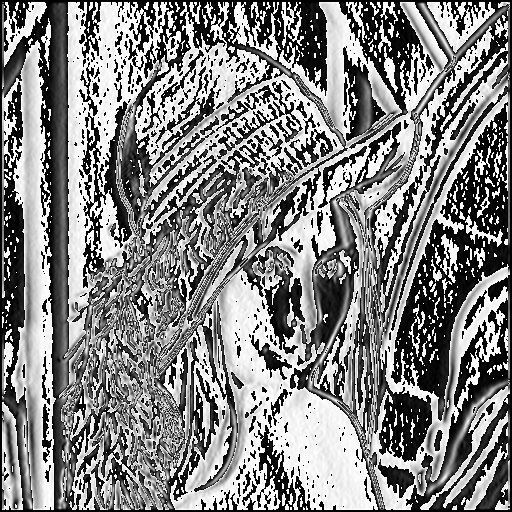
**Compute Horizontal/Vertical Gradient:**

Gradient is the first-order derivatives of image for each direction. It is rough detection of all possible edges in the image and the edges look thick. So we need to thinning algorithm to find 1-pixel edge lines, which is Non-Maximal Suppression. The gradient can be computed using central difference:

deltaX(x,y) = [(x+1, y) - (x-1, y)] / 2

deltaY(x,y) = [(x, y+1) - (x, y-1)] / 2

Horizontal Gradient: Vertical Gradient:

### **Compute Magnitude of the Gradient**

Magnitude of horizontal and vertical gradient is used for non-maximal suppression process. The magnitude can be computed by:

magnitude = sqrt(deltaX\*deltaX + deltaY\*deltaY)



### **Non-Maximal Suppression (NMS):**

NMS is an edge-thinning algorithm. This process results in one pixel wide ridges from thick edges. It requires h/v gradients and the magnitude of gradients. The basic idea of NMS is that: If a pixel value is not greater than its neighbored pixels, then the pixel is not the edge (the scalar value of the pixel is set to zero). During this process, it also needs to know the direction of the gradient vectors in order to find 2 neighbour pixels on the same direction, then to compare them with the current pixel. (mostly 4 or 8 directions are required).



### **Perform Hysteresis Threshold**

Hysteresis thresholding is used to remove the weak edges and plus, connect the splitted edges. To connect edges, starting at a pixel which is greater than high threshold and search all 8 surround neighbours. If the neighbour is greater than low threshold, then it will also become an edge. The range of threshold is: 0 < tLow < tHigh < 1

**Applications:**

Edge detection is one of the fundamental steps in

Image processing,

Image analysis,

Image pattern recognition, and

Computer vision techniques.

**References:**

Bmp Loading:

<http://www.vbforums.com/showthread.php?t=261522>

<http://en.wikipedia.org/wiki/BMP_file_format>

Bmp Padding:

<http://en.wikipedia.org/wiki/BMP_file_format#Pixel_storage>

Gaussian Filter:

<http://en.wikipedia.org/wiki/Canny_edge_detector>

<http://fourier.eng.hmc.edu/e161/lectures/canny/node1.html>

<http://www.tomgibara.com/computer-vision/CannyEdgeDetector.java>

Scheduling:

<https://msdn.microsoft.com/en-us/library/9w1x7937.aspx>

**Section 2:**

The serial code is a step wise implementation of the processes. We can parallelize few steps for a speed up.The complexity of the algorithm is **O(mn log mn)** where m and n are dimensions of the given image. The complexity is mainly determined by the convolution operation, whose order is NlogN where N is the number of times the convolution operation is to be performed.

The three steps which did not have any kind of dependency and hence were parallelizable were: Gradient calculation, non-maximal suppression and edge-tracking by hysteresis. The gaussian-filter procedure is inherently serial in nature.

Gradient calculation: The loops vary across all the pixels in the image and gradient is calculated and stored in an array of pixels.

Non-maximal suppression: The loops vary across all the pixels of the image and for each pixel, the direction of the gradient is checked for. All the edges in which the current pixel is the maxima are retained,otherwise they are ignored.

Edge-tracking by hysteresis: The loops again vary across all the pixels and a path is found out by checking for neighbouring pixels’ brightness. Corresponding edge is created in the output image.

The effect of increasing the problem size (or image size) increases the time taken by the serial code as the procedures individually have process a lot on the image. We noted that there are more number of convolution operations when the problem size increases for a fixed sigma value (the computation time is dominated by the convolution procedure), thus the procedure became very extensive computationally. Also, more neighbouring pixels have to be checked for in case of gradient,non-max suppression and edge-tracking.

**Section 3:**

The parallelization strategy aims at reducing the time taken to complete the data-independent task, by inducing some concurrency.

Gradients are calculated for each pixel individually.

Weak edges are computed in relation to each pixel.

Edges are tracked by taking into account the neighbouring pixels of each pixel individually.

Thus these three sections are parallelizable. Thus pixels are handed out to the threads,which process concurrently over different pixels to produce the output image matrix.

We have used static scheduling ,dynamic scheduling with a chunk-size of eight and guided scheduling for the same.

**Section 4:**

**Results:**

Input Image: Output Image:

Sigma=1.0 and kernel Size =7 Sigma=2.0 and kernel Size =11

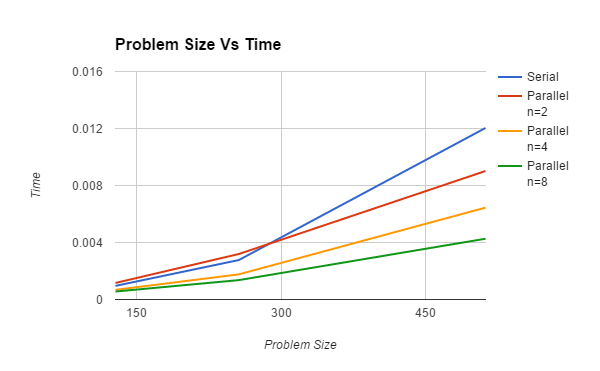
 

**Observations:**

***All the observations are taken for sigma=1.0 and kernel Size =7.***

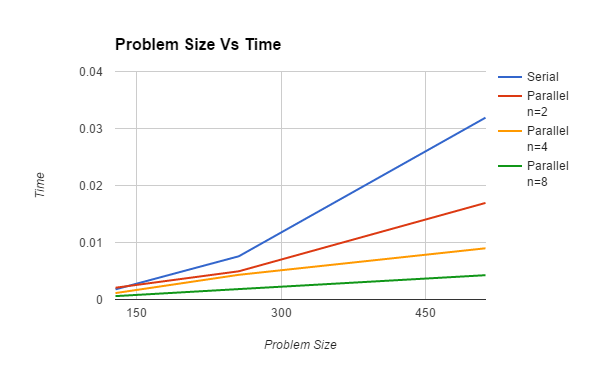
Problem Size VS Time for Gradient:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem Size | Serial | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| 128 | 0.000956 | 0.001161 | 0.000678 | 0.000560 |
| 256 | 0.003192 | 0.002768 | 0.001764 | 0.001360 |
| 512 | 0.012036 | 0.009016 | 0.006446 | 0.004264 |



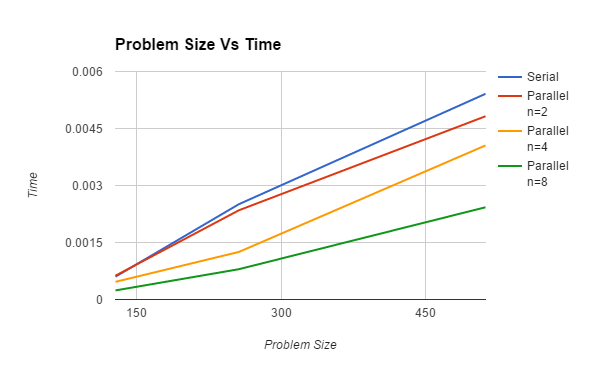
Problem Size VS Time for Non-Maximal Suppression:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem Size | Serial | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| 128 | 0.001781 | 0.002064 | 0.001126 | 0.000592 |
| 256 | 0.007584 | 0.004951 | 0.004332 | 0.001827 |
| 512 | 0.0319 | 0.016942 | 0.008983 | 0.004264 |



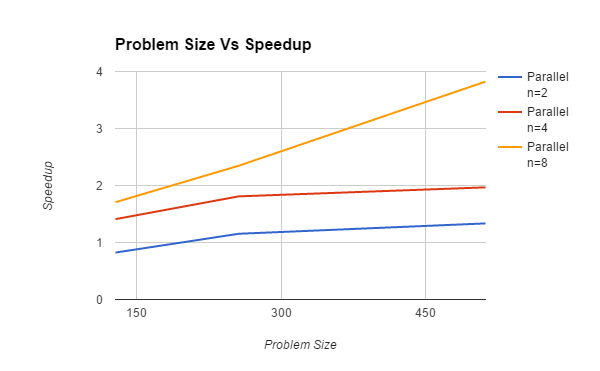
Problem Size VS Time for Hysteresis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem Size | Serial | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| 128 | 0.000601 | 0.000627 | 0.000465 | 0.000239 |
| 256 | 0.002507 | 0.002347 | 0.001251 | 0.000797 |
| 512 | 0.005412 | 0.004822 | 0.004054 | 0.002427 |



Problem Size VS Speedup for Gradient:

|  |  |  |  |
| --- | --- | --- | --- |
| Problem Size | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| 128 | 0.823422 | 1.410029 | 1.707142 |
| 256 | 1.153179 | 1.809523 | 2.347058 |
| 512 | 1.33496 | 1.967204 | 3.822701 |



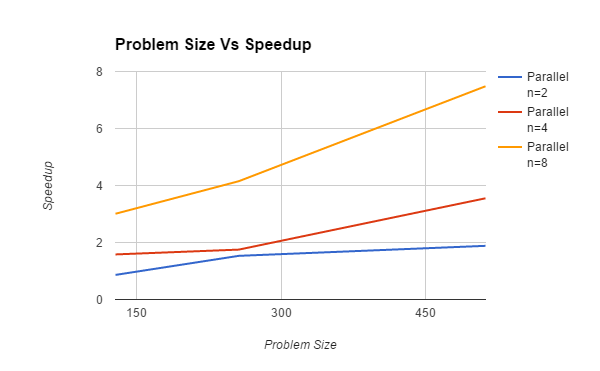
The speed up for small problem-size is lesser than 1 - owing to the synchronization overheads. Parallel slowdown is happening in this case where the benefit due to parallelization is lesser than the overheads involved.

For a fixed problem size, with increase in the number of cores, the speed up increases, however we note that the efficiency decreases.

For fixed number of cores, the speedup increases with increase in problem size since the effect of overhead on the speedup decreases as there is more scope for parallelization.

Problem Size VS Speedup for Non-Maximal Suppression:

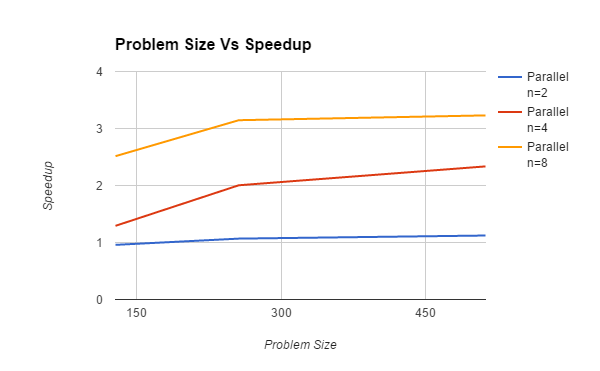
|  |  |  |  |
| --- | --- | --- | --- |
| Problem Size | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| 128 | 0.862887 | 1.581705 | 3.008445 |
| 256 | 1.531811 | 1.750692 | 4.151067 |
| 512 | 1.882894 | 3.551152 | 7.481238 |



Again, similar trends of speed up are followed by the procedure of non-maximal suppression.

Problem Size VS Speedup for Hysteresis

|  |  |  |  |
| --- | --- | --- | --- |
| Problem Size | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| 128 | 0.958553 | 1.292473 | 2.514644 |
| 256 | 1.068172 | 2.003996 | 3.145545 |
| 512 | 1.122355 | 2.334977 | 3.229913 |



The speedup varies in a similar manner with respect to the problem-size and the number of cores as was in the case for gradient-finding, since the scheduling and computation patterns are similar.

**Karp-Flatt Metric:**

The fraction of serial code experimentally determined in image of size 512:

e=(1/-1/p)/1-1/p

Gradient:

|  |  |  |  |
| --- | --- | --- | --- |
| Value of e | n=2 : 0.49857 | n=4 : 0.444517 | n=8 : 0.46359 |

Non-Maximal Suppression

|  |  |  |  |
| --- | --- | --- | --- |
| Value of e | n=2 : 0.06219 | n=4 : 0.06209 | n=8 : 0.05996 |

Hysteresis:

|  |  |  |  |
| --- | --- | --- | --- |
| Value of e | n=2 : 0.68205 | n=4 : 0.68771 | n=8 : 0.61097 |

In Non-maximal suppression major percentage of the code is parallel implementation, so we see that speedup is nearly equal to that of number of processors.

We see that e almost remains constant for all the procedures, thus there is inherently serial portion in each of these parallel parts. However overheads have the least effect on non-maximal suppression since e is very less.

**Scheduling:**

All the observations are taken for the image of size 512x512.

Gradient Calculation:

|  |  |  |  |
| --- | --- | --- | --- |
| Schedule | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| Static | 0.009016 | 0.006446 | 0.004264 |
| Guided | 0.009000 | 0.005124 | 0.004200 |
| Dynamic | 0.014701 | 0.008663 | 0.005964 |

Non-Maximal Suppression:

|  |  |  |  |
| --- | --- | --- | --- |
| Schedule | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| Static | 0.016942 | 0.008983 | 0.004264 |
| Guided | 0.015993 | 0.008640 | 0.004019 |
| Dynamic | 0.016894 | 0.009089 | 0.004539 |

Hysteresis:

|  |  |  |  |
| --- | --- | --- | --- |
| Schedule | Parallel n=2 | Parallel n=4 | Parallel n=8 |
| Static | 0.004822 | 0.004054 | 0.002427 |
| Guided | 0.004669 | 0.003976 | 0.002028 |
| Dynamic | 0.005049 | 0.004909 | 0.0039 |

The time taken for each procedure shows that guided works better than static and dynamic scheduling. This is because of the fact that the guided scheduling is characterized by the fewest number of synchronizations .

Static scheduling works better than dynamic scheduling because when adjacent pixels are processed, cache efficiency would be ensured, since the chunk-size is predefined. The data would already exist in the cache.

In dynamic scheduling, since the pixels to be processed are randomly handed out, there would be lot of synchronization overheads.

**Section 5:**

The major observations is with increase in the kernel size the image after gaussian filter gets smoother and smoother, which causes removal of few useful edges, hence the value of sigma can not be huge.

The output image after serial implementation and parallel implementation differ a little due the distribution of work between threads.

Further improvement can be done by implementing the code for a careful image as we have done it only for a grayscale image.

Also we have tried but not succeeded in parallelising the gaussian filter properly, so a perfect parallel implementation of the filter can be done.