**Non Local Means (NLM) De-noising algorithm**

1. Description

NLM algorithm is used in image processing applications for removing the noise. The main objective of the image de-noising algorithm is to remove the unwanted signal while preserving the original image as much as possible. Compared to other de-noising algorithms, Non Local Means Algorithm for de-noising helps preserve the edges and finer details of the image. This algorithm was proposed by Buades et. al. This project aims to implement this algorithm in parallel using GPU and serially using CPU and compare the performances and profiling information and to measure the Speedup.

V(i)=U(i)+N(i)

where V(i) is the observed image, U(i) is the true image and N(i) is the noise at a pixel 𝑖.

Given a discrete noisy image v = {v(i) | i ∈ I}, the estimated value NL[v](i), for a pixel i, is computed as a weighted average of all the pixels in the image,

NL[v](i)

where, the family of weights {w(i, j)}j depend on the similarity between the pixels i and j.

The similarity between two pixels i and j depends on the similarity of the intensity gray level vectors v(Ni) and v(Nj), where Nk denotes a square neighborhood of fixed size and centered at a pixel k. Buades et. al. calls this square neighborhood, a similarity window or a patch.

For computational purposes of the NL-means algorithm, we can restrict the search of similar windows in a larger ”search window” of size S×S pixels. A search window size of 21×21 pixels and a similarity square neighborhood Ni of 7×7 pixels is assumed. The pixels outside the search window do not contribute to the value of the output. This property allows us to separate the image into independent disjoint pieces and process them in parallel.

Buades et. Al. calculated the weights associated which each pixel is computed using the formula given below:

e−(𝑖,𝑗)/ℎ2

where 𝑍(𝑖) is the normalizing constant and ssd refers to the sum of squared deviations. Z(i) is the sum of all weights and is used to normalize the weight of each patch. h is the degree of filtering.

Method\_noise helps us to understand the performance and limitations of the denoising algorithms. If the NL-means method noise does not present any noticeable geometrical structures, it mean that the filter performance is good.

1. **Algorithm**
2. Calculate the optimum work group size configuration from Kernel Development Framework.
3. In the host side: Convert RGB input image to Greyscale image grey scale image.
   1. Employing the Luminosity method, convert the RGB values to the corresponding grey values.
4. Generate a Gaussian white noise distribution Gaussian distribution with mean = 0 and standard deviation = 2.
5. Add the generated Gaussian distribution in step 2 to all pixels of the output of step 1 to generate noisy image.
6. Denoising algorithm: Group the image into patches 7 x 7 pixels called similarity windows. The selected pixel must be at the center of the similarity window. Group these patches into search window of size 21 x 21 pixels.
7. Calculate the Euclidean distance at each similarity window within the search window. Euclidean distance is the sum of squared deviations from each pixel t all of the neighboring pixels within the search window.
8. Calculate the sum of the Euclidean distance, which is the normalizing constant.
9. Assigns weight values depending on the Euclidean distance and filtering parameter.
10. For each pixel, calculate the sum of product of Euclidean distances and the weights of the neighboring pixels.
11. The de-noised image value at a given pixel is the value of value calculated in step 8 normalized to the sum of the weights that is the normalizing constant.
12. Repeat steps 1 to 9 for each pixel.
13. Calculate the execution time, serial execution time Windows Performance Counters (WPC).
14. Send the input Image, image width, height, number of channels, sampler used and the filtering parameter to the device side.
15. Implement separate kernels to calculate the grey scale image*,* noisy image *and the* denoised Image. Repeat steps 1 to 9 in Device side and perform the Parallel implementation.
16. Generate the method noise image to find out the effectiveness of the filtered output. This image is the difference between the input image and the filtered output.
17. Use the serial and parallel execution times to calculate the efficiency, speed up, serial fraction and OPI.
18. **OpenCL Parallel Design**

* First the input image is converted into a grey scale image. Conversion of RGB values to grey values are done on separate work items
* Additive White Gaussian noise is added to the image.
* Computation of the weights associated with each pixel is done on a separate work item. The different work items working in parallel at any time, reduces the computation time drastically.
* After the computation of weights for each pixel, the pixel value is modified.

The modified pixel values are then written to an output image which is the de-noised version of the input image

1. **Pseudocode**

**Serial implementation**

For all pixels in the inputImage repeat the following steps.

float luminance = 0.2126\*input\_image.x + 0.7152\* input\_image.y + 0.0722\* input\_image.z;

The output greyScaleImage will have R,G,B values equal to luminence.

Generate a Gaussian noise distribution using the built in function of C++ with a mean of 0 and Standard deviation = 2 as follows:

* + - normal\_distribution<float> gaussianDistribution(0.0f, 1.0f);

Add the Gaussian distribution values at each pixel of greyScaleImage to generate a noisyImage.

For each pixel in the 21x21 search window

* + For each pixel p in the 7x7 similarity window
    - Euclidean distance = sum f squared deviations between pixel p and neighbor pixel
  + end for

Update the weight depend on the Euclidean distances and filtering parameter h.

Calculate the averageVal = average of weights \* grey value at each pixel

Calculate denoisedImage which is the averageVal denoised to the sum of weights.

End for

Calculate the execution time, *serialTime* Windows Performance Counters (WPC).

End for

method\_noise function implementation

For each pixel of the inputImg

method\_noise = inputImg – outputIg

end for

**Parallel implementation pseudocode**

1. Calculate the optimum work group size configuration from Kernel Development Framework.
2. Send the *inputImg*, image width, height, number of channels, sampler and the filtering parameter to the device side.
3. convert\_to\_greyscale kernel

for each pixel of inputImg

float luminance = 0.2126\*input\_image.x + 0.7152\* input\_image.y + 0.0722\* input\_image.z;

greyScaleImage.x = greyScaleImage.y = greyScaleImage.z = luminance

1. add\_noise kernel

Generate a gaussianNoiseDistribution using with a mean of 0 and Standard deviation = 2.

For each pixel of the inputImg image

noisyImage = gaussianNoiseDistribution + greyScaleImage

end for

1. denoise kernel

for each pixel of inputImg

For each pixel in the 21x21 search window

For each pixel in the 7x7 similarity window

Euclidean distance = sum of squared deviations between pixel p and neighbor pixel

End for

Update the weight depend on the Euclidean distances and filtering parameter h.

Calculate the averageVal = average of weights \* grey value at each pixel

Calculate denoisedImage which is the averageVal

End for

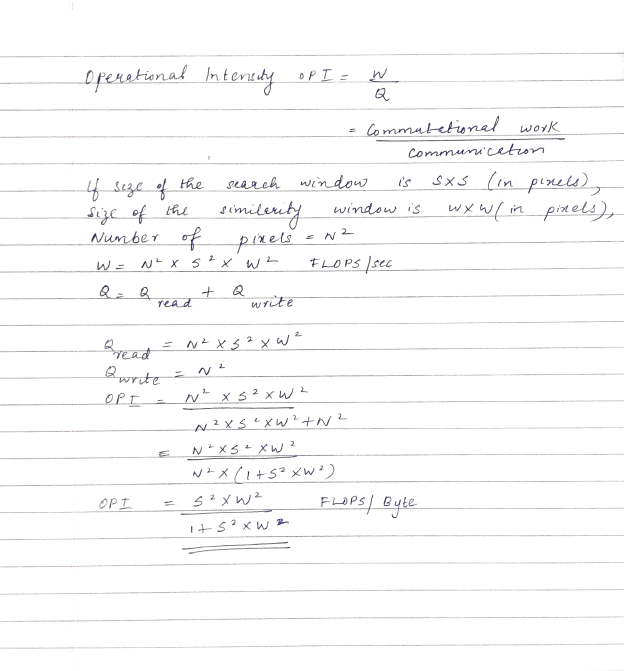
outputImg is obtained by denoising to the sum of weights.

1. method\_noise kernel

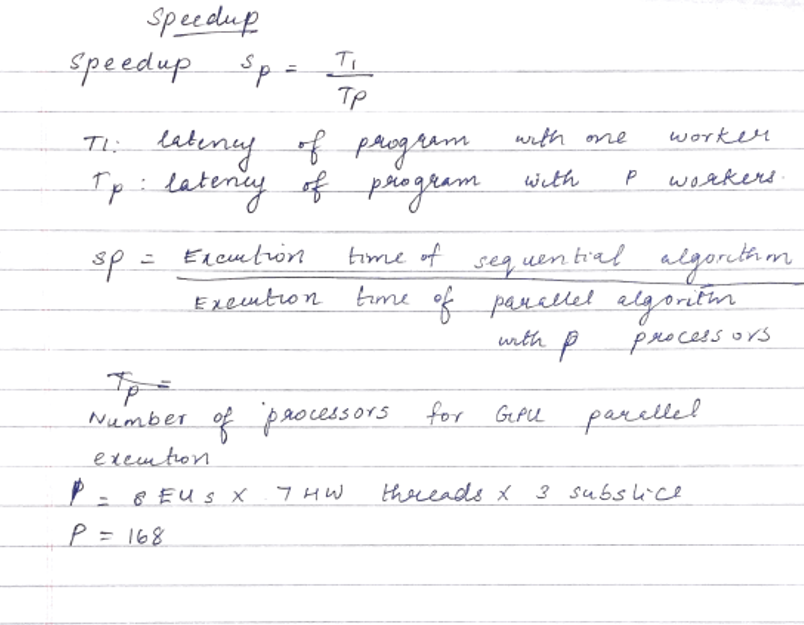
For each pixel of the inputImg

method\_noise = inputImg – outputImg

1. **Theoretical analysis**



* Theoretical speedup



* Theoretical Efficiency

𝑓 be the non-parallelizable serial fraction of the total work

* Theoretical serial fraction fe

Wser = T1f

* Peak floating point performance = 806.4 GFLOPS/ sec

1. **Measured analysis**
2. Operational Intensity

OPI =

Operational Intensity is approximately equal to 0.99.

1. Speed up (calculated in Table 1)
2. Roofline Performance Model

Figure 1 Roofline Performance Model

According to VTune, FPU utilization is 25.6% = .256 x 806.4

= 206.43 GFLOPS/ sec

1. **Profiling results**

Table 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Function Implemented | Windows Performance Counter execution time(ms) | | Windows Performance Counter Standard deviation (ms) | | Speed Up | OpenCL profiling execution time(ms) | OpenCL profiling Standard deviation (ms) |
| Parallel | Serial | Parallel | Serial |
| Convert to Greyscale (512x512) | 0.730000 (greyscale.cl) | 22.503000 | 0.451381 | 1.371549 | 30.82 | 0.263248 | 0.029334 |
| Generate noisy image(512x512) |  | 26.218 |  | 0.653709 |  |  |  |
| Denoise (512x512) | 5445.041992 | 50325820.000000 | 11.887500 | 39892.992188 | 9242 | 62151336.000000 | 37107.710938 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

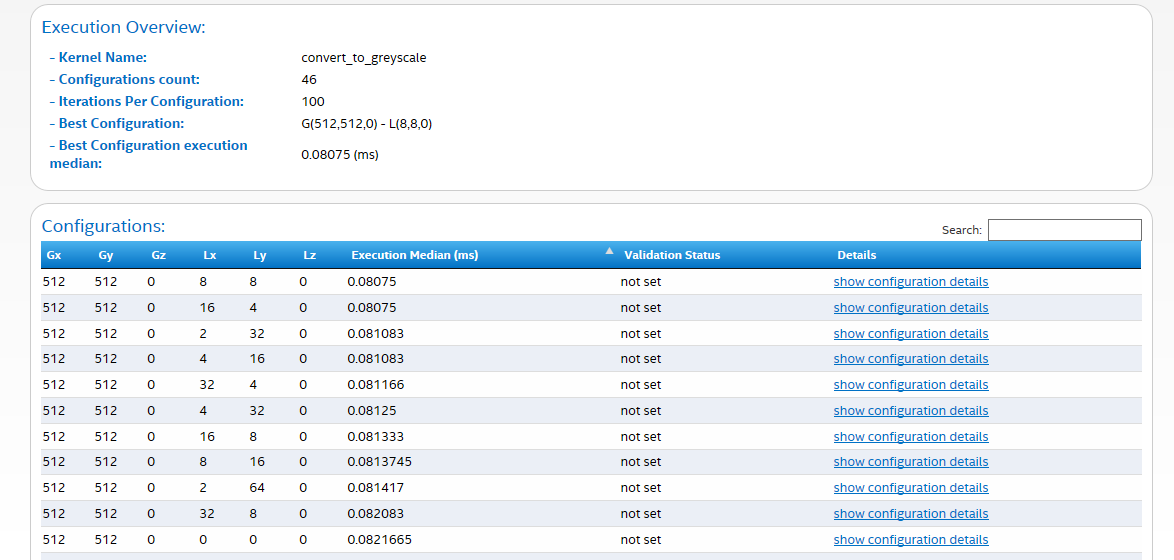
1. **Performance results from VTunes**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Kernel | Execution Unit Utilization | | | EU Threads Occupancy | 2 FPUs active | IPC Rate | Untyped Memory Bandwidth, GB/sec | |
| Active | Stalled | Idle | Read | Write |
| Denoise\_algorithm | 71.6 % | 28.4 % | 0.0 % | 99.5 % | 25.7 % | 1.387 | 0 | 0 |
| Convert\_to\_greyscale | 36 % | 4.8 % | 59.2 % | 36.6 % | 25.0 % | 1.712 | 0.038 | 1.165 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Kernel | Shared Local Memory Bandwidth, GB/sec | | Typed Memory Bandwidth, GB/sec | | GPU Memory Bandwidth, GB/sec | | L3 shader bandwidth GB/ sec | Sampler | |
| Read | Write | Read | Write | Read | Write | Busy | Bottleneck | |
| Denoise\_algorithm | 0 | 0 | 0 | 0 | 0.002 | 0.009 | 0.365 | 100.0% | 0.6% | |
| Convert\_to\_greyscale | 0 | 0 | 0.057 | 0.155 | 1.399 | 0.402 | 1.284 | 36.8% | 0.9% | |

1. **Discussion of results**

* Command to run the .exe - EmptyOpenCLProject2.exe cat.png denoised.jpg
* Work group size was optimized using KDF for 100 iterations. Optimum configuration of work group size is (8,8,0).



* Outputs

Greyscale image

Parallel implementation



Serial implementation



Noisy image

Serial implementation



Denoised Image

Parallel implementation



Serial implementation



Method noise



1. **References:**

* NLM by Buades et. al.
* GPU Based Fast Non Local Means Algorithm by Daniel Sanju Antony and G. N. Rathna Electrical Engineering, Indian Institute of Science (IISc), Bangalore, India
* <https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/>
* <https://dsp.stackexchange.com/questions/29522/what-does-it-mean-by-adding-gaussian-noise-to-a-2d-image>