Project Name: Image Histogram by barraCUDA team

Team Members: Vincent Vilda, Jacob Osorio

Demo and Presentation Video: https://youtu.be/7NwBPWKMiY0

https://youtube.com/shorts/jcrfhiBUHvQ?si=eBhAU3TMlZPtjviE

Project Idea / Overview

The main idea of this project is to perform image histogram calculations using a GPU inorder to increase performance. Four different methods of calculating the histogram are performed. Histogram on the CPU is conducted by using an OpenCV histogram function, while three different kernels will be utilized on the GPU with an incrementing method of optimization. The elapsed times are taken for comparison for each of the four methods.

Project Description: The goal of this project is to develop an image histogram computation using CUDA and compare its performance to a similar implementation using OpenCV. The histogram of an image is a graphical representation of the tonal distribution in a digital image. By utilizing GPU acceleration through CUDA, we aim to significantly speed up the computation process compared to OpenCV histogram function, which typically runs on a CPU. The first method used on the GPU is using a kernel with a block dimension of 1x1. The second method is using a block dimension of 32x32. The last method is by using a kernel with a block dimension of 32x32 and shared local bins.

How is the GPU used to accelerate the application?

Parallel Algorithm/CUDA Reduction Algorithm: A reduction algorithm will be implemented to sum up pixel values efficiently.

Problem Space Partitioning: Threads and Thread Blocks: The image will be divided into chunks (tiling), with each chunk assigned a number of threads. Multiple threads will be grouped into thread blocks to take advantage of the GPU architecture.

Data Parallelism: The task of calculating the histogram will be parallelized across the image's pixels, allowing for simultaneous computation.

Data Loading: Load image data into GPU memory.

Kernel Execution: Execute CUDA kernels for reduction and histogram computation.

Result Aggregation: Combine results atomically (atomicAdd) from each thread block into the final histogram. Shared memory for local binning was also used.

Comparison: Execute a similar pipeline in openCV for performance comparison against GPU computing.

Implementation Details

Data Transfer: Transfer image data to GPU memory using cudaMemcpy.

Kernel Functions: Write CUDA kernels for reduction and histogram computation. The first method uses a 1x1 block dimension. The second method uses a 32x32 block dimension. The third method uses a shared memory for local bin and also a 32x32 block dimension.

Memory Management: Efficiently manage GPU memory to avoid bottlenecks. Reduced use of _syncthred().

openCV Implementation: The Image is uploaded using OpenCV. The image is uploaded as black and white. Image features such as, dimension, and channel can be used. The image is then processed for histogram calculation using an OpenCV function calcHist( ).

Image Processing Toolbox: Utilize openCV built-in functions calcHist( ) for histogram computation.

Performance Measurement: Time the execution of the histogram computation.

Comparison: The time execution of the OpenCV on the CPU was much faster than the ones calculated by the GPU. This could be due to the size of the image. The larger the image the better the result would be for the GPU. The bigger block dimension on the GPU also demonstrated faster time.

Metrics: Measure execution time and compare the performance between CUDA and openCV implementations.

Analysis: Analyze the reasons for performance differences. The poorer performance of the GPU is due to its overhead cost. For it to perform better the input should have a much larger element count.

Documentation

git@github.com:VincentVilda/Final-Project.git

<u>Evaluation/Results</u>

Performance Metrics: Time taken for histogram computation in both CUDA and openCV . Used three histogram kernels for testing trials with the amount of time elapsed.

Analysis: Time elapsed was slower for a GPU histogram than with a CPU(OpenCV histogram).

Visualization: Graphs comparing execution times for different image sizes and types.

<u>Problems Faced</u>

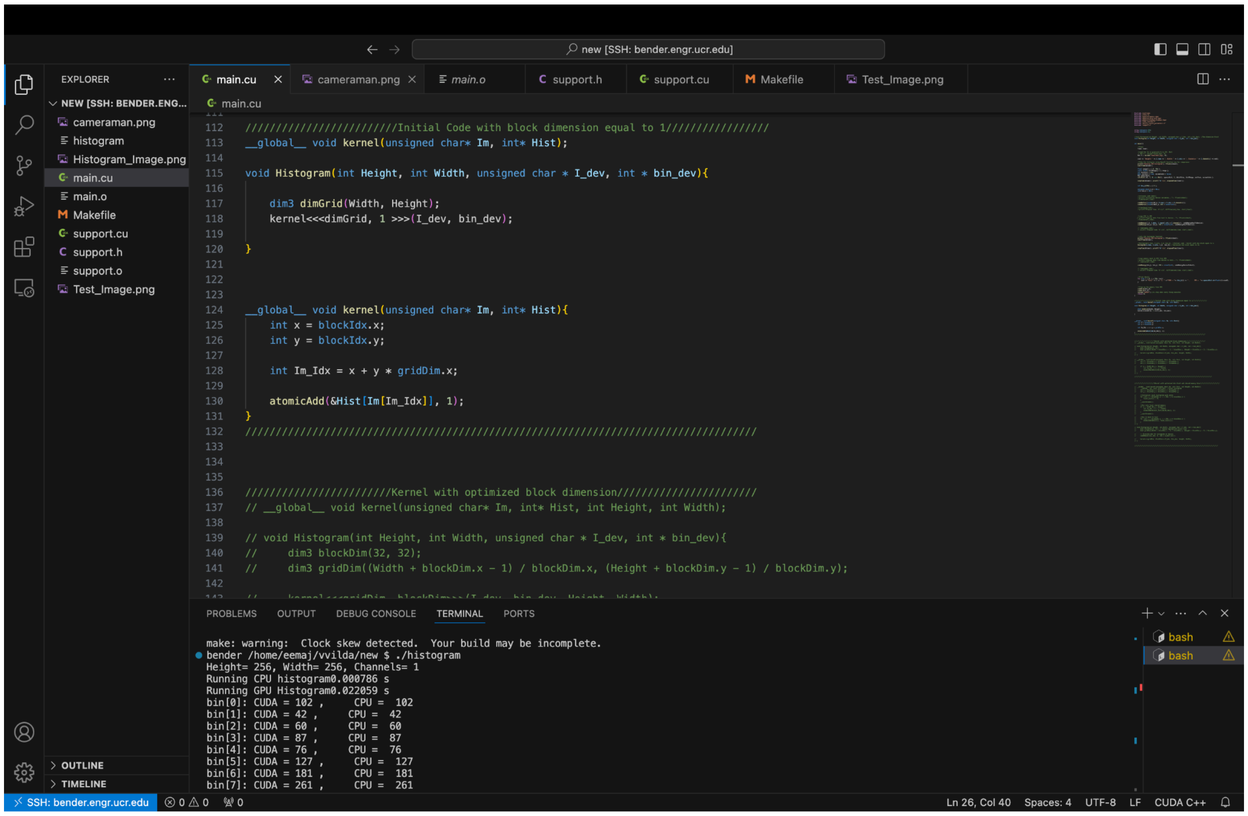
CUDA - Issues such as debugging parallel code, managing memory efficiently.

openCV - Issues: Optimizing openCV code for comparison and using proper library functions.

General Issues: Handling non square images was not tested, however the kernel with a dimension block of 1x1 would be able to perform on samples that are not square. Ensuring accurate performance measurement required the use of precise timing. Initially the timer.h was used but was conflicting with the results. A support code written by the Professor was used to accurately get the elapsed time for each performance.

Task	GPU histogram	CPU (OpenCV histogram Function)
Three Histogram Kernels used for Comparison	**Time elapsed (s)**	**Time elapsed (s)**
Kernel with 1x1 Block Dim	0.022059	0.000786
Kernel with 32x32 Block Dim	0.011056	0.000668
Kernel with 32x32 Block Dim and Share Memory Local Bin	0.011214	0.000847

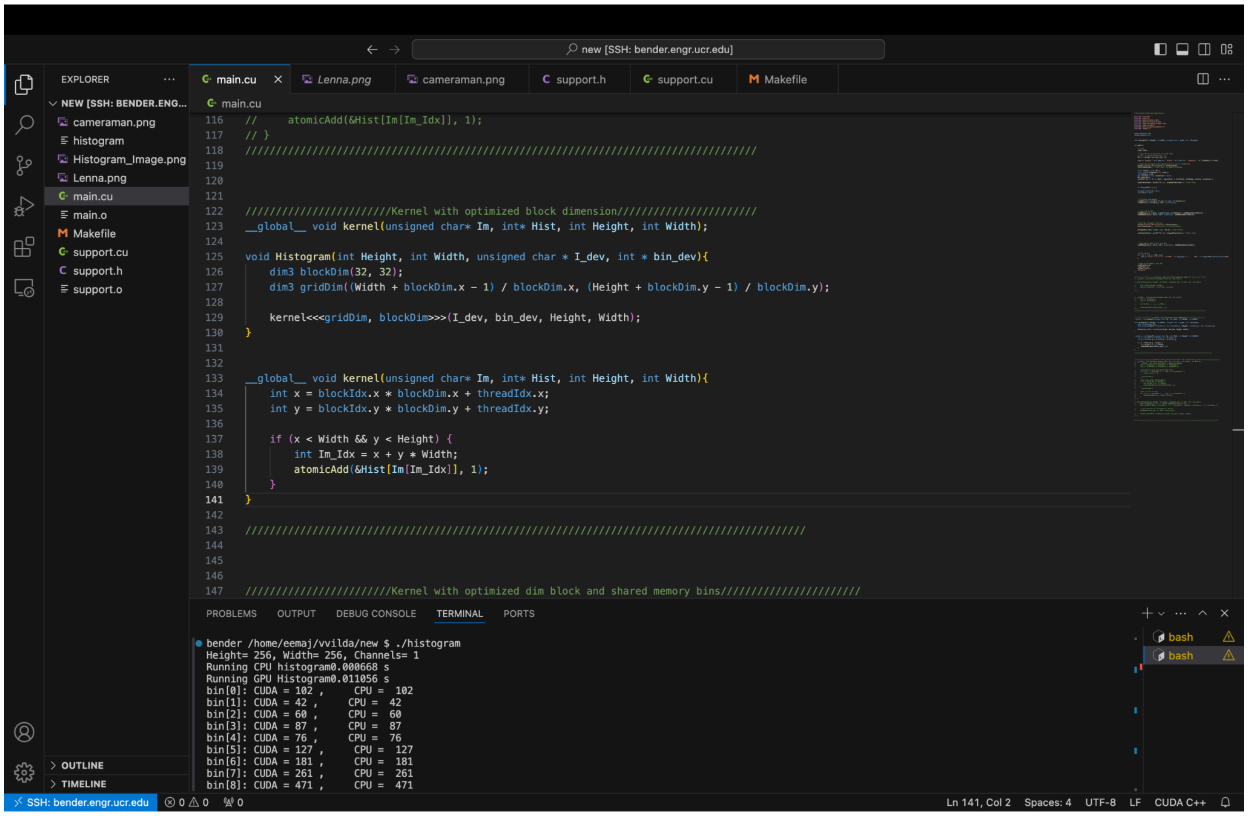
The figure below shows a kernel with 1x1 block dimension used for gpu histogram. The result shows that the CPU is still much faster in producing the histogram.



```
//////////////////////////Initial Code with block dimension equal to 1/////////////////
__global__ void kernel(unsigned char* Im, int* Hist);

void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){

    dim3 dimGrid(Width, Height);
    kernel<<<dimGrid, 1 >>>(I_dev, bin_dev);

}



__global__ void kernel(unsigned char* Im, int* Hist){
    int x = blockIdx.x;
    int y = blockIdx.y;

    int Im_Idx = x + y * gridDim.x;

    atomicAdd(&Hist[Im[Im_Idx]], 1);
}
////////////////////////////////////////////////////////////////////////////////////



////////////////////////Kernel with optimized block dimension//////////////////////
// __global__ void kernel(unsigned char* Im, int* Hist, int Height, int Width);

// void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
//     dim3 blockDim(32, 32);
//     dim3 gridDim((Width + blockDim.x - 1) / blockDim.x, (Height + blockDim.y - 1) / blockDim.y);

```

```
make: warning:  Clock skew detected.  Your build may be incomplete.
bender /home/eemaj/vvilda/new $ ./histogram
Height= 256, Width= 256, Channels= 1
Running CPU histogram0.000786 s
Running GPU Histogram0.022059 s
bin[0]: CUDA = 102 ,     CPU =  102
bin[1]: CUDA = 42 ,     CPU =  42
bin[2]: CUDA = 60 ,     CPU =  60
bin[3]: CUDA = 87 ,     CPU =  87
bin[4]: CUDA = 76 ,     CPU =  76
bin[5]: CUDA = 127 ,     CPU =  127
bin[6]: CUDA = 181 ,     CPU =  181
bin[7]: CUDA = 261 ,     CPU =  261
```

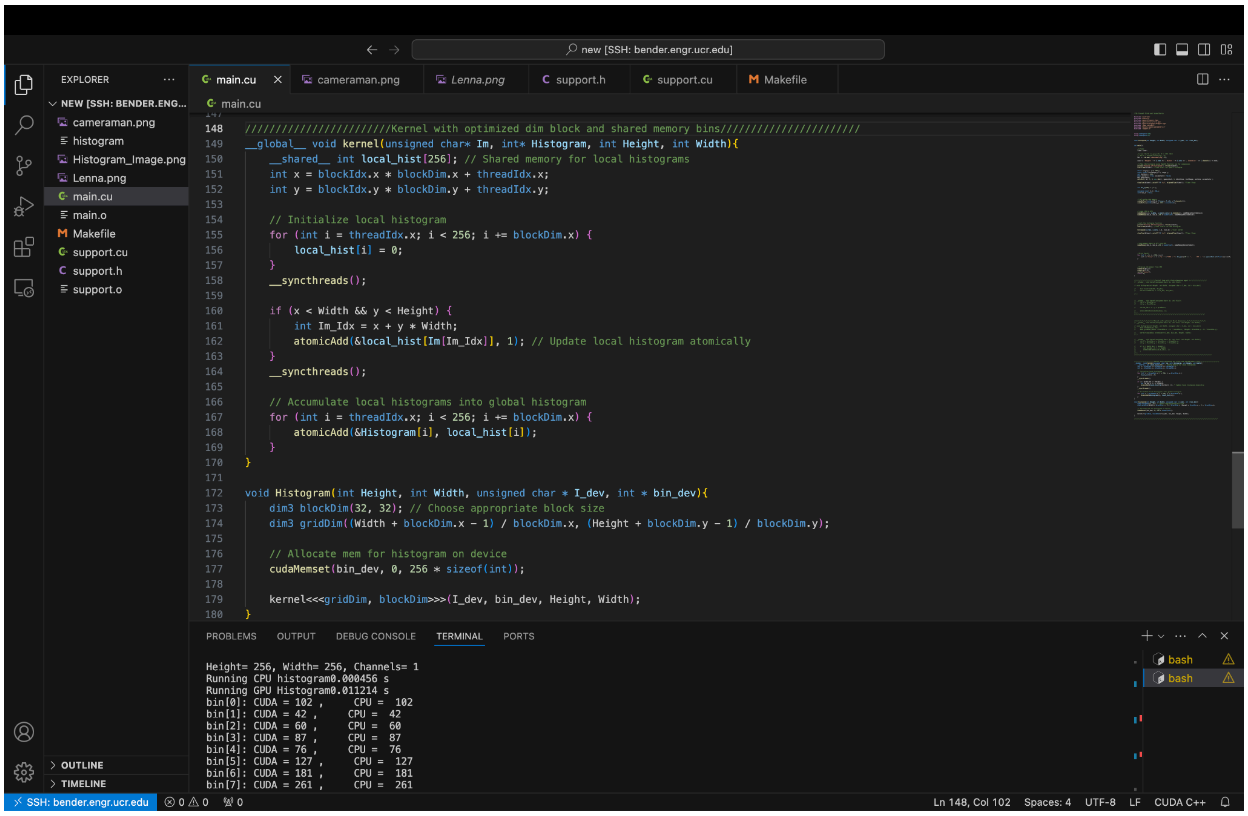
The figure below shows a kernel with 32x32 block dimension used for gpu histogram. The result shows that the CPU is still much faster in producing the histogram. The result also shows that this is faster than the kernel with a block dimension of only 1x1.



```
//     atomicAdd(&Hist[Im[Im_Idx]], 1);
// }
/////////////////////////////////////////////////////////////////////////////////////



///////////////////////Kernel with optimized block dimension/////////////////////
__global__ void kernel(unsigned char* Im, int* Hist, int Height, int Width);

void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
    dim3 blockDim(32, 32);
    dim3 gridDim((Width + blockDim.x - 1) / blockDim.x, (Height + blockDim.y - 1) / blockDim.y);

    kernel<<<gridDim, blockDim>>>(I_dev, bin_dev, Height, Width);
}


__global__ void kernel(unsigned char* Im, int* Hist, int Height, int Width){
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x < Width && y < Height) {
        int Im_Idx = x + y * Width;
        atomicAdd(&Hist[Im[Im_Idx]], 1);
    }
}

/////////////////////////////////////////////////////////////////////////////////////////////



///////////////////////Kernel with optimized dim block and shared memory bins/////////////////////
```

```
bender /home/eemaj/vvilda/new $ ./histogram
Height= 256, Width= 256, Channels= 1
Running CPU histogram0.000668 s
Running GPU Histogram0.011056 s
bin[0]: CUDA = 102 ,     CPU =  102
bin[1]: CUDA = 42 ,     CPU =  42
bin[2]: CUDA = 60 ,     CPU =  60
bin[3]: CUDA = 87 ,     CPU =  87
bin[4]: CUDA = 76 ,     CPU =  76
bin[5]: CUDA = 127 ,     CPU =  127
bin[6]: CUDA = 181 ,     CPU =  181
bin[7]: CUDA = 261 ,     CPU =  261
bin[8]: CUDA = 471 ,     CPU =  471
```

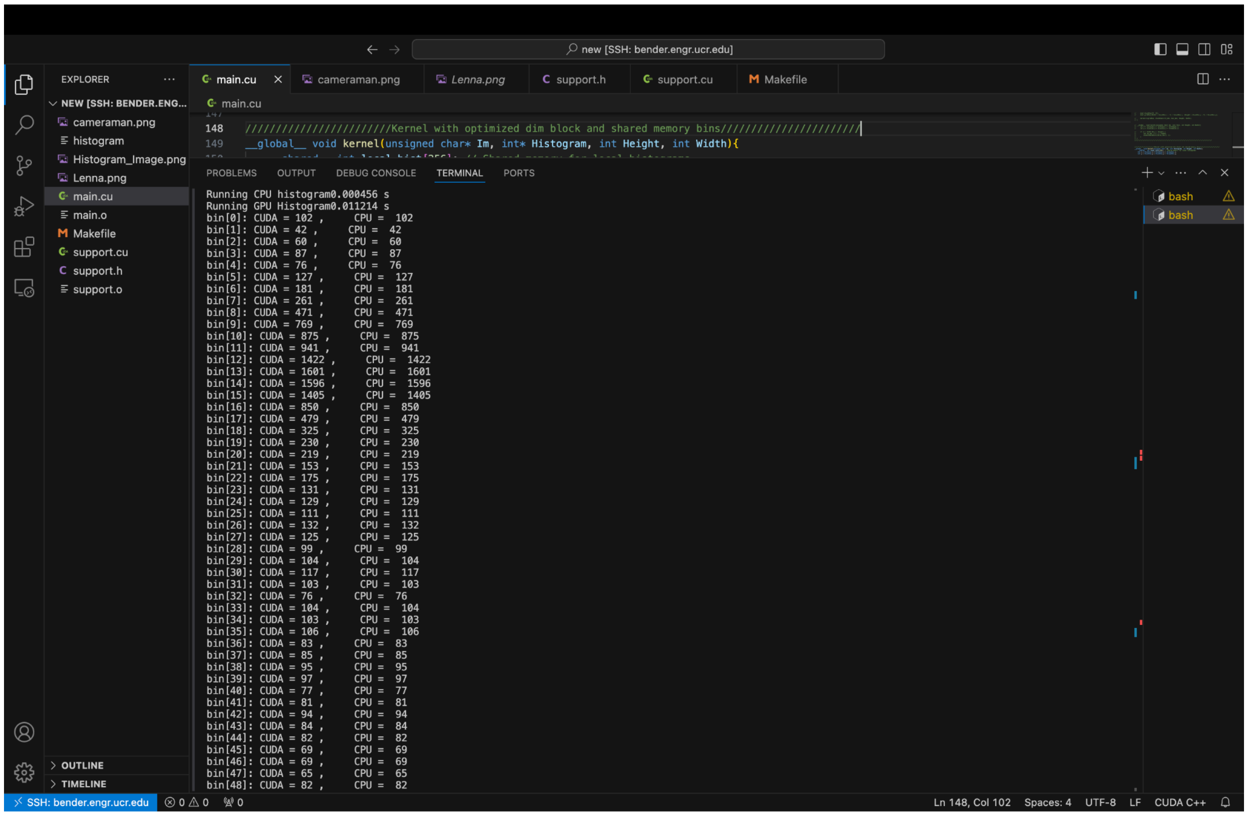
The figure below shows a kernel with shared local bin and 32x32 block dimension used for gpu histogram. The result shows that the CPU is still much faster in producing the histogram. The result also shows that this is faster than the kernel with a block dimension of only 1x1, and almost similar in speed as to the kernel with 32x32 blockdimesion and no shared local bin.



```
///////////////////////Kernel with optimized dim block and shared memory bins//////////////////////
__global__ void kernel(unsigned char* Im, int* Histogram, int Height, int Width){
    __shared__ int local_hist[256]; // Shared memory for local histograms
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    // Initialize local histogram
    for (int i = threadIdx.x; i < 256; i += blockDim.x) {
        local_hist[i] = 0;
    }
    __syncthreads();

    if (x < Width && y < Height) {
        int Im_Idx = x + y * Width;
        atomicAdd(&local_hist[Im[Im_Idx]], 1); // Update local histogram atomically
    }
    __syncthreads();

    // Accumulate local histograms into global histogram
    for (int i = threadIdx.x; i < 256; i += blockDim.x) {
        atomicAdd(&Histogram[i], local_hist[i]);
    }
}

void Histogram(int Height, int Width, unsigned char * I_dev, int * bin_dev){
    dim3 blockDim(32, 32); // Choose appropriate block size
    dim3 gridDim((Width + blockDim.x - 1) / blockDim.x, (Height + blockDim.y - 1) / blockDim.y);

    // Allocate mem for histogram on device
    cudaMemset(bin_dev, 0, 256 * sizeof(int));

    kernel<<<gridDim, blockDim>>>(I_dev, bin_dev, Height, Width);
}
```

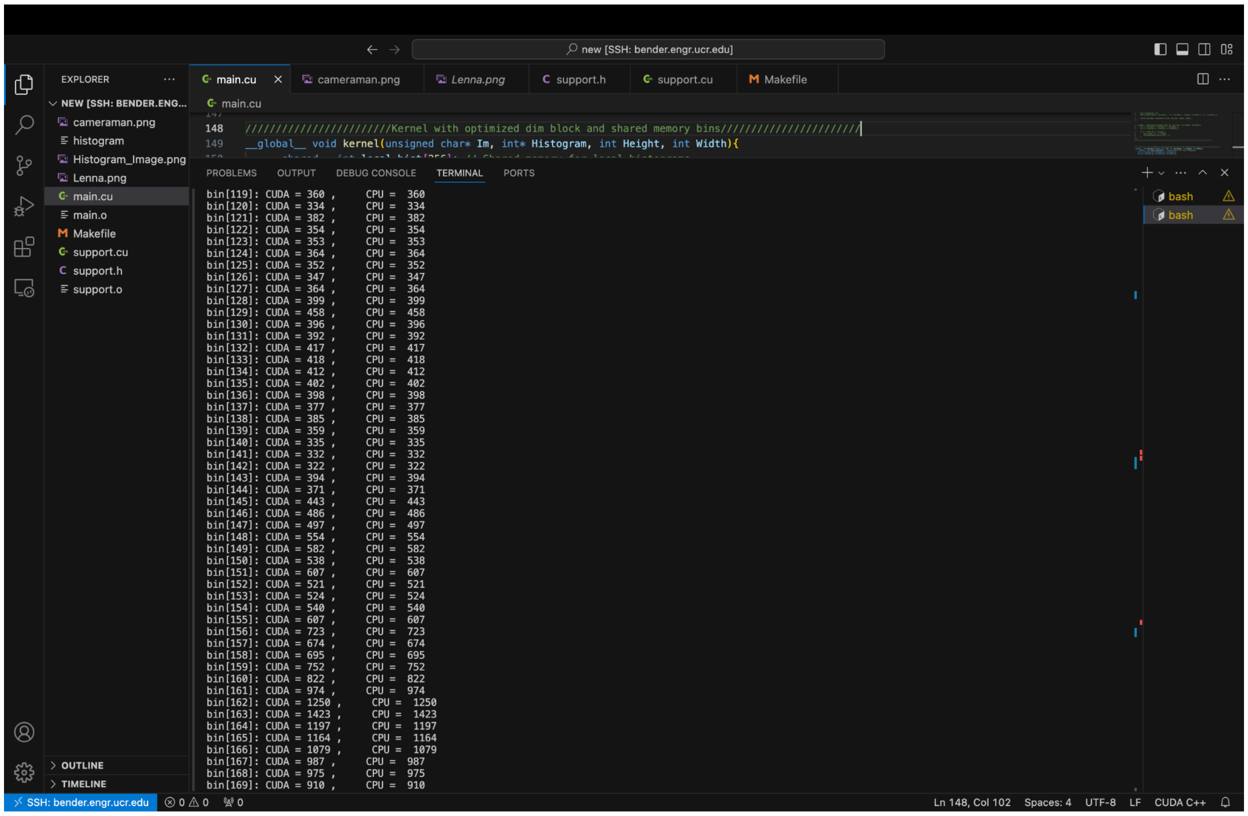
```
Height= 256, Width= 256, Channels= 1
Running CPU histogram0.000456 s
Running GPU Histogram0.011214 s
bin[0]: CUDA = 102 ,     CPU =  102
bin[1]: CUDA = 42 ,     CPU =  42
bin[2]: CUDA = 60 ,     CPU =  60
bin[3]: CUDA = 87 ,     CPU =  87
bin[4]: CUDA = 76 ,     CPU =  76
bin[5]: CUDA = 127 ,     CPU =  127
bin[6]: CUDA = 181 ,     CPU =  181
bin[7]: CUDA = 261 ,     CPU =  261
```

The two images below show the result of the histogram computed by the GPU as compared to the one that is calculated by the OpenCV function in the CPU.



```
////////////////////////Kernel with optimized dim block and shared memory bins/////////////////////
__global__ void kernel(unsigned char* Im, int* Histogram, int Height, int Width){
```

```
Running CPU histogram0.000456 s
Running GPU Histogram0.011214 s
bin[0]: CUDA = 102 ,     CPU =  102
bin[1]: CUDA = 42 ,     CPU =  42
bin[2]: CUDA = 60 ,     CPU =  60
bin[3]: CUDA = 87 ,     CPU =  87
bin[4]: CUDA = 76 ,     CPU =  76
bin[5]: CUDA = 127 ,     CPU =  127
bin[6]: CUDA = 181 ,     CPU =  181
bin[7]: CUDA = 261 ,     CPU =  261
bin[8]: CUDA = 471 ,     CPU =  471
bin[9]: CUDA = 769 ,     CPU =  769
bin[10]: CUDA = 875 ,     CPU =  875
bin[11]: CUDA = 941 ,     CPU =  941
bin[12]: CUDA = 1422 ,     CPU =  1422
bin[13]: CUDA = 1601 ,     CPU =  1601
bin[14]: CUDA = 1596 ,     CPU =  1596
bin[15]: CUDA = 1405 ,     CPU =  1405
bin[16]: CUDA = 850 ,     CPU =  850
bin[17]: CUDA = 479 ,     CPU =  479
bin[18]: CUDA = 325 ,     CPU =  325
bin[19]: CUDA = 230 ,     CPU =  230
bin[20]: CUDA = 219 ,     CPU =  219
bin[21]: CUDA = 153 ,     CPU =  153
bin[22]: CUDA = 175 ,     CPU =  175
bin[23]: CUDA = 131 ,     CPU =  131
bin[24]: CUDA = 129 ,     CPU =  129
bin[25]: CUDA = 111 ,     CPU =  111
bin[26]: CUDA = 132 ,     CPU =  132
bin[27]: CUDA = 125 ,     CPU =  125
bin[28]: CUDA = 99 ,     CPU =  99
bin[29]: CUDA = 104 ,     CPU =  104
bin[30]: CUDA = 117 ,     CPU =  117
bin[31]: CUDA = 103 ,     CPU =  103
bin[32]: CUDA = 76 ,     CPU =  76
bin[33]: CUDA = 104 ,     CPU =  104
bin[34]: CUDA = 103 ,     CPU =  103
bin[35]: CUDA = 106 ,     CPU =  106
bin[36]: CUDA = 83 ,     CPU =  83
bin[37]: CUDA = 85 ,     CPU =  85
bin[38]: CUDA = 95 ,     CPU =  95
bin[39]: CUDA = 97 ,     CPU =  97
bin[40]: CUDA = 77 ,     CPU =  77
bin[41]: CUDA = 81 ,     CPU =  81
bin[42]: CUDA = 94 ,     CPU =  94
bin[43]: CUDA = 84 ,     CPU =  84
bin[44]: CUDA = 82 ,     CPU =  82
bin[45]: CUDA = 69 ,     CPU =  69
bin[46]: CUDA = 69 ,     CPU =  69
bin[47]: CUDA = 65 ,     CPU =  65
bin[48]: CUDA = 82 ,     CPU =  82
```



```
///////////////////////Kernel with optimized dim block and shared memory bins/////////////////////
__global__ void kernel(unsigned char* Im, int* Histogram, int Height, int Width){
```

```
bin[119]: CUDA = 360 ,      CPU =  360
bin[120]: CUDA = 334 ,      CPU =  334
bin[121]: CUDA = 382 ,      CPU =  382
bin[122]: CUDA = 354 ,      CPU =  354
bin[123]: CUDA = 353 ,      CPU =  353
bin[124]: CUDA = 364 ,      CPU =  364
bin[125]: CUDA = 352 ,      CPU =  352
bin[126]: CUDA = 347 ,      CPU =  347
bin[127]: CUDA = 364 ,      CPU =  364
bin[128]: CUDA = 399 ,      CPU =  399
bin[129]: CUDA = 458 ,      CPU =  458
bin[130]: CUDA = 396 ,      CPU =  396
bin[131]: CUDA = 392 ,      CPU =  392
bin[132]: CUDA = 417 ,      CPU =  417
bin[133]: CUDA = 418 ,      CPU =  418
bin[134]: CUDA = 412 ,      CPU =  412
bin[135]: CUDA = 402 ,      CPU =  402
bin[136]: CUDA = 398 ,      CPU =  398
bin[137]: CUDA = 377 ,      CPU =  377
bin[138]: CUDA = 385 ,      CPU =  385
bin[139]: CUDA = 359 ,      CPU =  359
bin[140]: CUDA = 335 ,      CPU =  335
bin[141]: CUDA = 332 ,      CPU =  332
bin[142]: CUDA = 322 ,      CPU =  322
bin[143]: CUDA = 394 ,      CPU =  394
bin[144]: CUDA = 371 ,      CPU =  371
bin[145]: CUDA = 443 ,      CPU =  443
bin[146]: CUDA = 486 ,      CPU =  486
bin[147]: CUDA = 497 ,      CPU =  497
bin[148]: CUDA = 554 ,      CPU =  554
bin[149]: CUDA = 582 ,      CPU =  582
bin[150]: CUDA = 538 ,      CPU =  538
bin[151]: CUDA = 607 ,      CPU =  607
bin[152]: CUDA = 521 ,      CPU =  521
bin[153]: CUDA = 524 ,      CPU =  524
bin[154]: CUDA = 540 ,      CPU =  540
bin[155]: CUDA = 607 ,      CPU =  607
bin[156]: CUDA = 723 ,      CPU =  723
bin[157]: CUDA = 674 ,      CPU =  674
bin[158]: CUDA = 695 ,      CPU =  695
bin[159]: CUDA = 752 ,      CPU =  752
bin[160]: CUDA = 822 ,      CPU =  822
bin[161]: CUDA = 974 ,      CPU =  974
bin[162]: CUDA = 1250 ,      CPU =  1250
bin[163]: CUDA = 1423 ,      CPU =  1423
bin[164]: CUDA = 1197 ,      CPU =  1197
bin[165]: CUDA = 1164 ,      CPU =  1164
bin[166]: CUDA = 1079 ,      CPU =  1079
bin[167]: CUDA = 987 ,      CPU =  987
bin[168]: CUDA = 975 ,      CPU =  975
bin[169]: CUDA = 910 ,      CPU =  910
```