CUDA Image Blur

The purpose of this lab is to implement an efficient image blurring algorithm for an input image. The image is represented as RGB float values. You will operate directly on the RGB float values and use a 5x5 Box Filter to blur the original image to produce the blurred image.

Question 1:

Edit the code in P9Q1 to perform the following:

- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory

Instructions about where to place each part of the code is demarcated by the //@@ comment lines.

In P9Q1.cu, remove the #include <bits/stdc++.h> line as we are using vs cpp solution. Include the two cuda runtime and launch parameters header files.

wb.h is a header file (and functions) for heterogeneous parallel programming work. Source: https://github.com/ashwin/coursera-heterogeneous

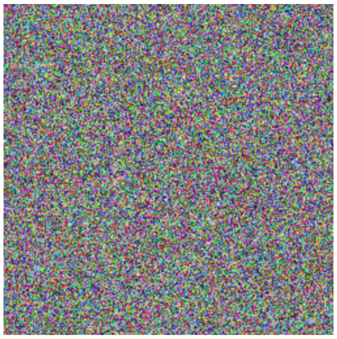
Input0.ppm is an input raster image stored in Portable PixMap format. To understand more about PPM format, refers to : RayTracing

To view PPM file (i.e., not able to be previewed in default Windows viewers), use: https://kylepaulsen.com/stuff/NetpbmViewer/

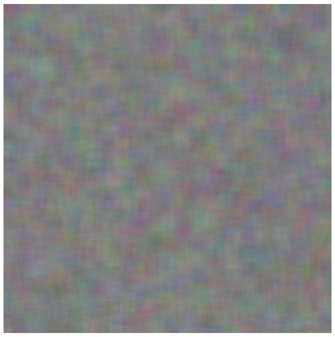
To execute the programme, we may need to provide ***two arguments***:
Path to “input0.ppm” as input, and path to “output0.ppm” as checking the transformed image is the same as output0.ppm file.

The generated file is called : **transformed_image.ppm**

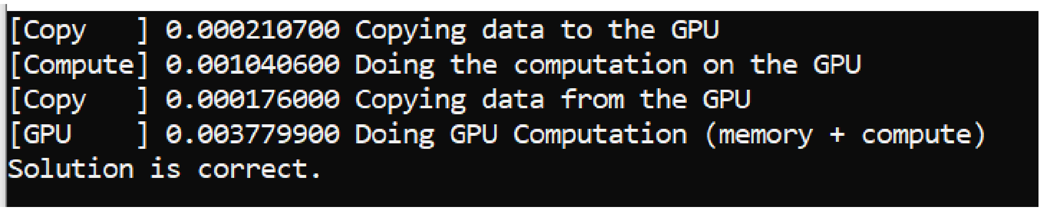
Input image:



Transformed image:



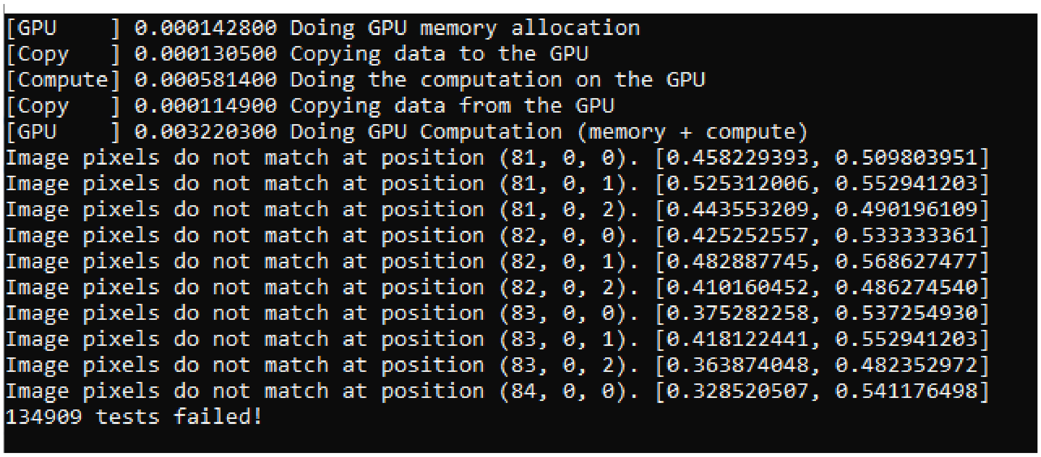
Output solution:



```
[Copy   ] 0.000210700 Copying data to the GPU
[Compute] 0.001040600 Doing the computation on the GPU
[Copy   ] 0.000176000 Copying data from the GPU
[GPU    ] 0.003779900 Doing GPU Computation (memory + compute)
Solution is correct.
```

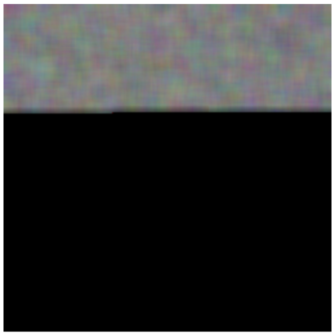
*** Example of wrong output:**

if we define the size as width * height * sizeof(float) , but we forgot the **channels**

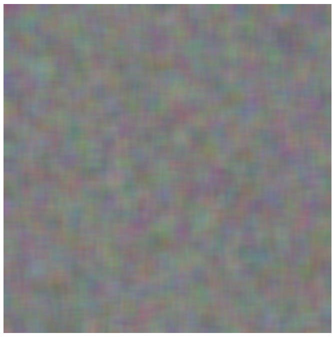


```
[GPU    ] 0.000142800 Doing GPU memory allocation
[Copy   ] 0.000130500 Copying data to the GPU
[Compute] 0.000581400 Doing the computation on the GPU
[Copy   ] 0.000114900 Copying data from the GPU
[GPU    ] 0.003220300 Doing GPU Computation (memory + compute)
Image pixels do not match at position (81, 0, 0). [0.458229393, 0.509803951]
Image pixels do not match at position (81, 0, 1). [0.525312006, 0.552941203]
Image pixels do not match at position (81, 0, 2). [0.443553209, 0.490196109]
Image pixels do not match at position (82, 0, 0). [0.425252557, 0.533333361]
Image pixels do not match at position (82, 0, 1). [0.482887745, 0.568627477]
Image pixels do not match at position (82, 0, 2). [0.410160452, 0.486274540]
Image pixels do not match at position (83, 0, 0). [0.375282258, 0.537254930]
Image pixels do not match at position (83, 0, 1). [0.418122441, 0.552941203]
Image pixels do not match at position (83, 0, 2). [0.363874048, 0.482352972]
Image pixels do not match at position (84, 0, 0). [0.328520507, 0.541176498]
134909 tests failed!
```

Transformed image



Provided Output image for checking



Histogram (Text)

A simple parallel histogram algorithm

- Partition the input into sections
- Have each thread to take a section of the input
- Each thread iterates through its section.
- For each letter, increment the appropriate bin counter

The purpose of this practice is to implement an efficient histogram algorithm for an input array of A-Z characters. There are 26 characters and each character will map into its own bin for a fixed total of 7 bins.

Question 2:

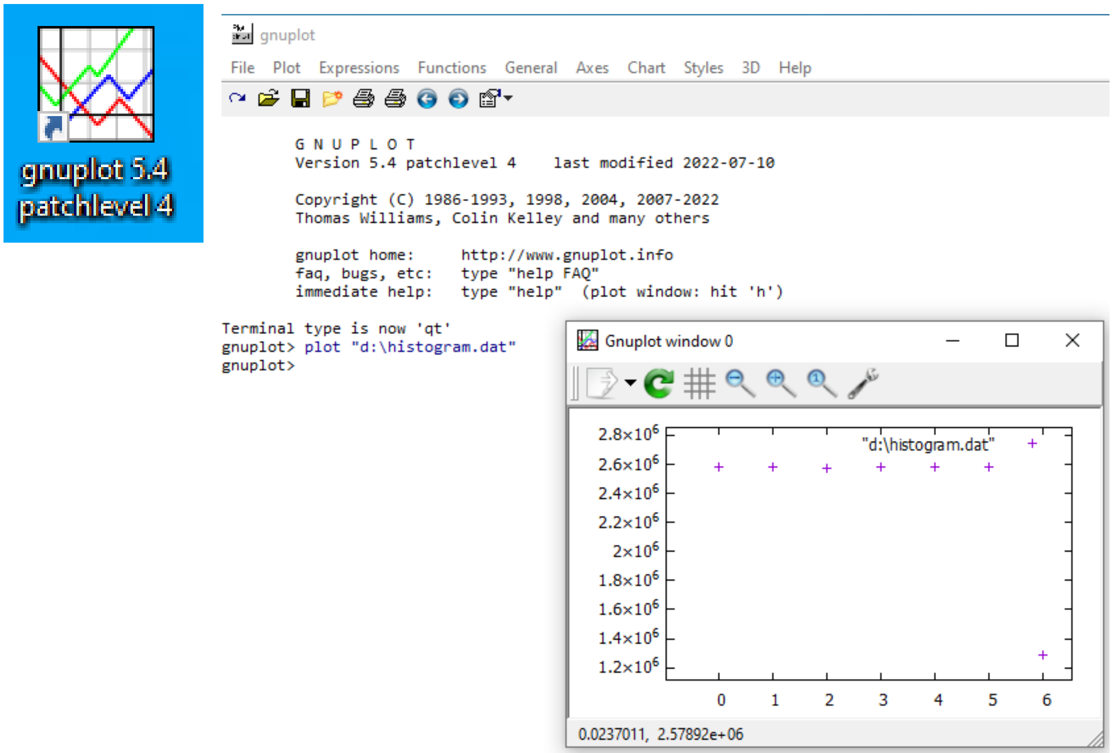
Use global memory allocation, execute the mapping of bins in parallel. Obtain the code in P9Q2.

- allocate device memory
- copy host memory to device (h_input.data(), h_result.data())
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory

Instructions about where to place each part of the code is demarcated by the //@@ comment lines.

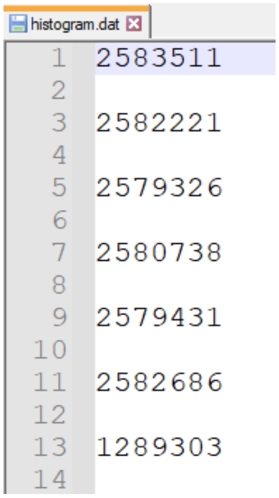
For thread block and kernel grid dimension definitions, refers to:
https://www.cs.ucr.edu/~amazl001/teaching/cs147/S21/slides/13-Histogram.pdf
Page 28.

The generated output histogram.dat can be opened by using gnuplot application, that can be downloaded from http://www.gnuplot.info/download.html. Launched the installed gnuplot and type plot ‘d:\histogram.dat’ to show the graph.



```
        G N U P L O T
        Version 5.4 patchlevel 4    last modified 2022-07-10

        Copyright (C) 1986-1993, 1998, 2004, 2007-2022
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot "d:\histogram.dat"
gnuplot>
```

Or, use a notepad (or notepad++) to open the histogram.dat



```
2583511

2582221

2579326

2580738

2579431

2582686

1289303

```

Question 3:

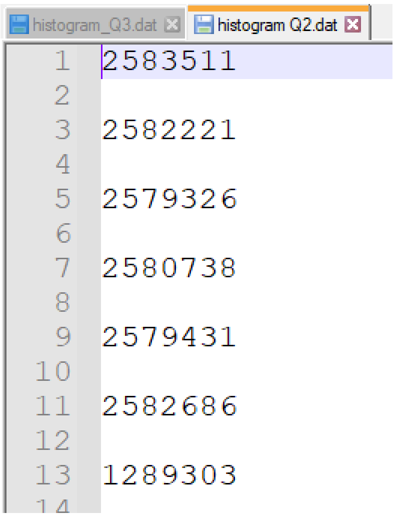
Use the approach of creating a privatized histogram in shared memory for each thread block, then atomically modifying the global histogram. Obtain the code P9Q3. Instructions about where to place each part of the code is demarcated by the
//@@ comment lines.

For **Additional understanding** of thread block and kernel grid dimension definitions, refers to:
https://safari.ethz.ch/projects_and_seminars/fall2021/lib/exe/fetch.php?media=p_s-hetsys-fs2021-meeting7-aftermeeting.pdf
Page 43

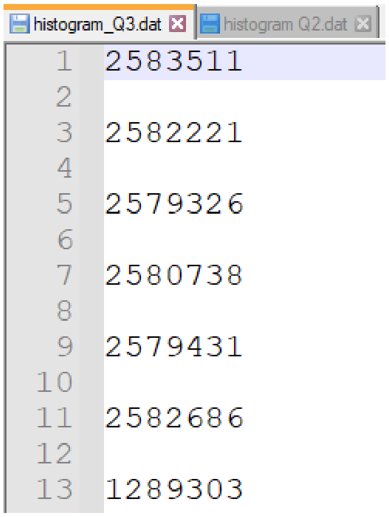
Use ***__syncthreads()*** to wait for shared memory writes to complete.

Add ***atomicAdd(&(s_result[alpha_position / DIV]), 1)*** after the line of
alpha_position = a[i] - 'a';

Check the results of Q3, should be same as Q2's results



```
2583511

2582221

2579326

2580738

2579431

2582686

1289303
```



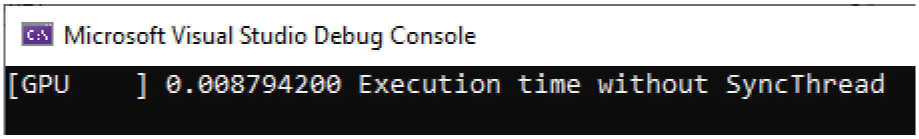
```
2583511

2582221

2579326

2580738

2579431

2582686

1289303
```

Question 4:

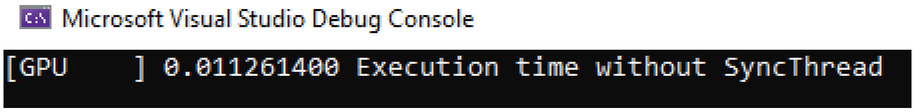
Compare the performance between Question 2 and Question 3 approaches in terms of computational time.

Tips:
Use wbTime_start and wbTime_stop from “wb.h”

Example result for Question 2:

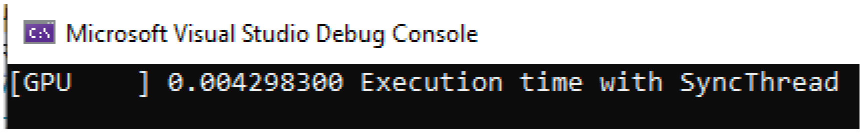


```
[GPU    ] 0.008794200 Execution time without SyncThread
```

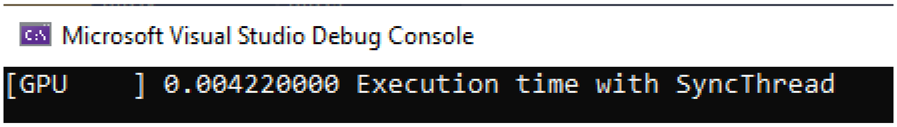


```
[GPU    ] 0.011261400 Execution time without SyncThread
```

Example result for Question 3:



```
[GPU    ] 0.004298300 Execution time with SyncThread
```



```
[GPU    ] 0.004220000 Execution time with SyncThread
```