HPC Lab4

Konstantinos Konsoulas
Ioannis Roumpos

18 December 2022

Contents

1 Introduction

The main goal of this assignment, is to choose an optimization strategy in order to reduce the execution time of a given sequential code when we transfer it in the GPU. In more detail, we are given the sequential code of a histogram equalization which is consisted from three parts. First, a histogram is calculated from a pgm-format image, then we calculate its cumulative distribution function to create a new histogram and finally a new equalized image is produced based on the histogram.

2 Parallel Patterns

To make a successful transfer of the histogram in the GPU, we should take into account that threads are going to access the same memory area many times. Thus, a combination of shared memory and atomic operations should be considered in order to avoid race conditions and have much less contention and serialization in accessing both private and final copy.

In the next phase, calculating the Cumulative Distribution Function of the histogram is a parallel scan problem and since the execution resources are adequate and we are interested in absolute performance we implemented the inefficient parallel scan algorithm also known as Kogge-Stone algorithm.

Lastly, the equalization of the image due to its "heavy" working is implemented in the GPU and to avoid multiple accesses in the global memory we saved the value of the array in a register to perform the calculations and then we copy it to the image array.

3 Failed Optimization

3.1 Tilling Technique

After profiling the GPU code ,as seen below, the **histogram** function takes the most execution time so we tried to implement tilling in this function. More specific, we split the memory in chunks of 8388608 bytes(8 MB) and call the kernel function as many times as the number of the chunks can fit in the image size. Although that may seem a good technique the overhead in the communication over the PCIe bus is big leading to bigger latency.

```
kkonsoulas@csl-artemis:~/lab4/Code$ nvprof ./gHist ../Images/planet_surface.pgm planet_surface.gpu
Running contrast enhancement for gray-scale images.
Image size: 6400 x 6400
==1778363== NVPROF is profiling process 1778363, command: ./gHist ../Images/planet_surface.pgm planet_surface.gpu
==1778363== Warning: Auto boost enabled on device 0. Profiling results may be inconsistent.
Starting GPU processing...
==1778363== Warning: Profiling results might be incorrect with current version of nvcc compiler used to compile cuda app. Compile with nvcc compiler 9.0 or later version to get corre
ct profiling results. Ignore this warning if code is already compiled with the recommended nvcc version
Elapsed time in GPU:  46.141342 ms
==1778363== Profiling application: ./gHist ../Images/planet_surface.pgm planet_surface.gpu
==1778363== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   72.44%  33.342ms         1  33.342ms  33.342ms  33.342ms  histogram(int*, unsigned char*, int, int)
                   27.47%  12.641ms         1  12.641ms  12.641ms  12.641ms  histogram_equalization(unsigned char*, unsigned char*, int*, int, int)
                    0.05%  23.807us         1  23.807us  23.807us  23.807us  histogram_prefixsum(int*, int*, int, int)
                    0.03%  14.112us         1  14.112us  14.112us  14.112us  histogram_calcdf(int*, int*, int)
                    0.01%  4.0320us         1  4.0320us  4.0320us  4.0320us  [CUDA memset]
      API calls:   67.26%  185.02ms         2  92.511ms  19.561ms  165.46ms  cudaHostAlloc
                   12.14%  33.398ms         3  11.133ms  20.301us  33.348ms  cudaDeviceSynchronize
                    9.96%  27.392ms         1  27.392ms  27.392ms  27.392ms  cudaDeviceReset
                    5.16%  14.186ms         2  7.0928ms  6.7313ms  7.4543ms  cudaFreeHost
                    4.60%  12.650ms         1  12.650ms  12.650ms  12.650ms  cudaEventSynchronize
                    0.38%  1.0327ms         2  516.37us  487.54us  545.20us  cuDeviceTotalMem
                    0.20%  540.99us       202  2.6780us     128ns  125.94us  cuDeviceGetAttribute
                    0.15%  402.55us         3  134.18us  2.8360us  394.75us  cudaMalloc
                    0.09%  252.64us         3  84.212us  2.9000us  239.73us  cudaFree
                    0.03%  73.367us         4  18.341us  5.6600us  52.559us  cudaLaunchKernel
                    0.02%  51.588us         2  25.794us  24.101us  27.487us  cuDeviceGetName
                    0.01%  21.827us         2  10.913us  3.9340us  17.893us  cudaEventRecord
                    0.01%  15.358us         1  15.358us  15.358us  15.358us  cudaMemset
                    0.01%  14.382us         2  7.1910us  1.3610us  13.021us  cudaEventCreate
                    0.00%  5.8210us         8     727ns     198ns  1.8730us  cudaGetLastError
                    0.00%  3.0520us         1  3.0520us  3.0520us  3.0520us  cudaEventElapsedTime
                    0.00%  1.7500us         3     583ns     227ns  1.2050us  cuDeviceGetCount
                    0.00%  1.4730us         4     368ns     206ns     662ns  cuDeviceGet
                    0.00%     477ns         2     238ns     182ns     295ns  cuDeviceGetUuid
kkonsoulas@csl-artemis:~/lab4/Code$
```

Figure 1: Nvidia profiler.

3.2 Constant Memory

Last but not least, we attempted to utilize the constant memory of the GPU with the aforementioned tilling technique but due to its minuscule size to just 64KB and its nature to be better utilized when read by all threads in the same warp at the same address makes it inefficient for our implementation.

4 Measurements

We tested the results of our implementation in all the images but in order to get times that are comparable we used the image planet surface for the measurements. In the graph we can see how the time varies depending the implementation.

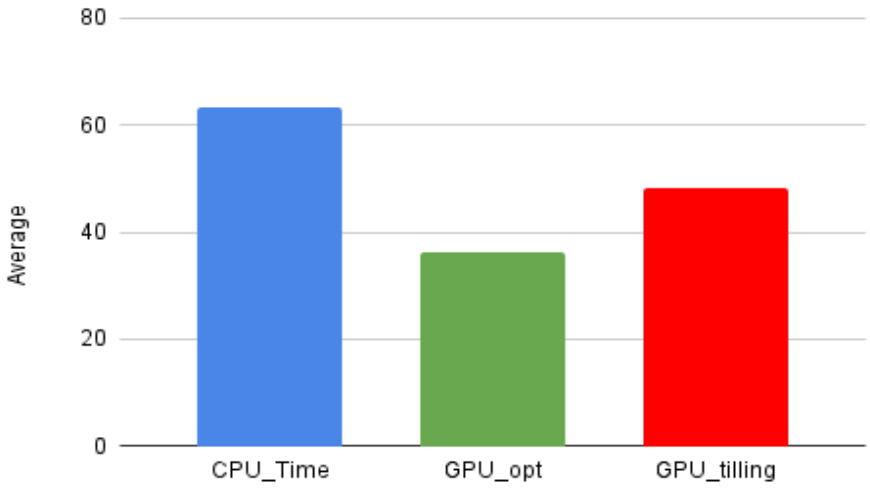

Figure 2: Time in milliseconds for different implementations.

5 Contact

Konstantinos Konsoulas(2975): kkonsoulas@uth.gr
Ioannis Roumpos (2980) : iroumpos@uth.gr