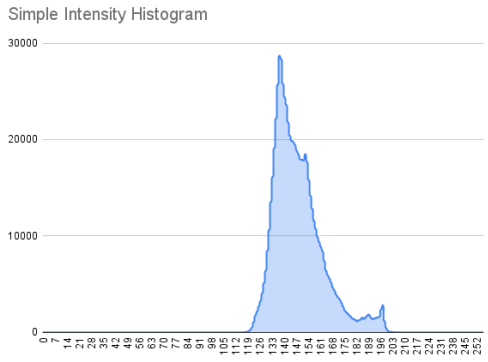
# CMP3752 Parallel Programming

Jacob Morgan — 25234606
25234606@students.lincoln.ac.uk

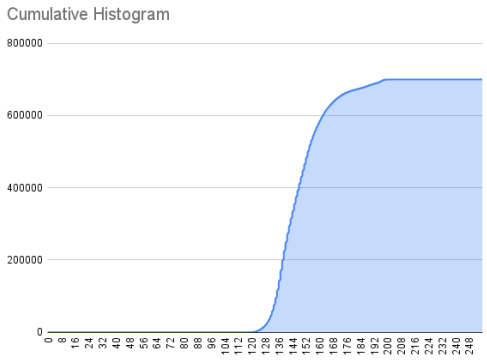
## Contents

# 1 Kernel Implementations

The basic implementation is taken from following the tutorials provided in the workshops (Wingate 2023). This includes an implementation working with 8-bit images, an intensity histogram and cumulative histogram implementation, which can be seen in figure 1. It also includes working read and write buffers for the kernel functions and profiling for each function.




(a) Intensity histogram on 8-bit grey scale image




(b) Cumlative histogram using inclusive hillis steele

Figure 1: Outputs from both histogram kernel functions

## 1.1 Intensity Histogram Implementation

My intensity histogram implementation is a parallel implementation of the histogram calculation which operates on global memory. This function operates using the *atomic_inc* function, which performs an increment operation on each work item based on the index of the bin into which the element falls. One issue with using atomic functions is that the atomic operation serialises access to global memory, making the process slower. To improve this function I could instead create a local barrier to ensure local memory is accessed.

## 1.2 Cumulative Histogram Implementations

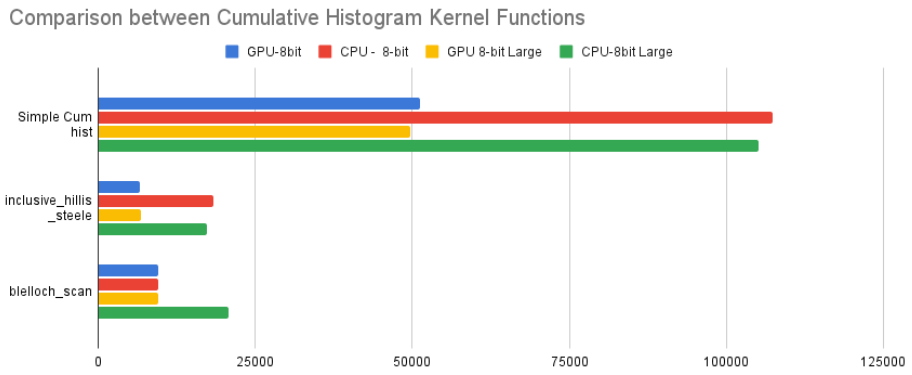



Figure 2: Cumulative Histogram Kernel Function Comparison

### 1.2.1 Simple Cumulative Histogram

This function is a standard cumulative sum function, which adds all elements and keeps all the results. To implement this, I used *atomic_add* which adds two values together, the two values being the current item selected in the workgroup and the item before.
This function, as seen in Figure 2, is the worst implementation as it takes the longest out of the three and its time complexity would be $O(n)$.

### 1.2.2 Inclusive Hillis-Steele

The inclusive Hillis-Steele function is a span-efficient kernel function that calculates the Cumulative Histogram in $\log_2(N)$ steps. This function forms a reduction tree for every output element and then merges the redundant elements of all the trees. As seen in Figure 2, Hillis-Steele performs best on the GPU but takes nearly twice as long on the CPU. This is because hardly any of the function is sequential (only the part in the for loop), the rest is fully parallel and runs on the GPU compute units.

### 1.2.3 Exclusive Blelloch Scan

This function has twice as many steps as the Hillis-Steele scan and three times more operations than the serial implementation. Blelloch scan has an up-sweep for reduction and a downward sweep to produce intermediate results. Blelloch scan has a span of $O(\log n)$ and is less efficient than Hillis-Steele. This pattern can also be seen in Figure 2.

### 1.3 Look Up Table Implementation

This kernel implements the histogram equalisation algorithm using a lookup table. Input $A$ contains the cumulative histogram and output $B$ contains an array for mapping pixel intensities to their equalised values. The time complexity of this kernel is $O(1)$ since each work item performs a single operation to compute the value for its corresponding bin.

### 1.4 Back-Projection Implementation

This is a fundamental step in the reconstruction of the image from the histogram. The input $A$ contains the image data and output $B$ contains the back-projected image. The kernel takes in the look-up table and maps the table values to the corresponding image pixels. The time complexity for this kernel function would be $O(1)$ as each work item performs a single lookup and assignment operation.

## 2 Colour Functionality

## 3 Variable Bins

## 4 Bit Depth Functionality

## 5 Platforms and Devices

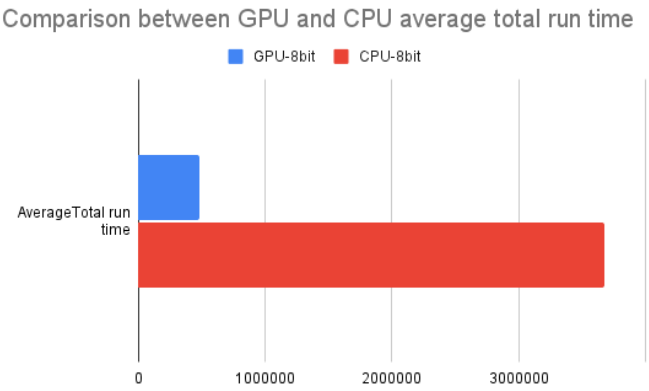



Figure 3

## References

Wingate, J. (2023), 'OpenCL Tutorials'. original-date: 2021-09-14T00:35:20Z.
**URL:** *https://github.com/wing8/OpenCL-Tutorials*