Advanced Computer Architecture (HPCA)
HW: GPU (CUDA) Histogram and Atomics

Overview: Use atomics and shared memory to compute a histogram of random number sequence

In this assignment, we will explore the use of atomics and shared memory - for this, we will write a CUDA program to compute a histogram of a random number sequence. A randomly generated list of *N* numbers between 0 and 9 is generated. The objective is to count how many numbers in the list are 0, how many are 1, how many are 2, etc.

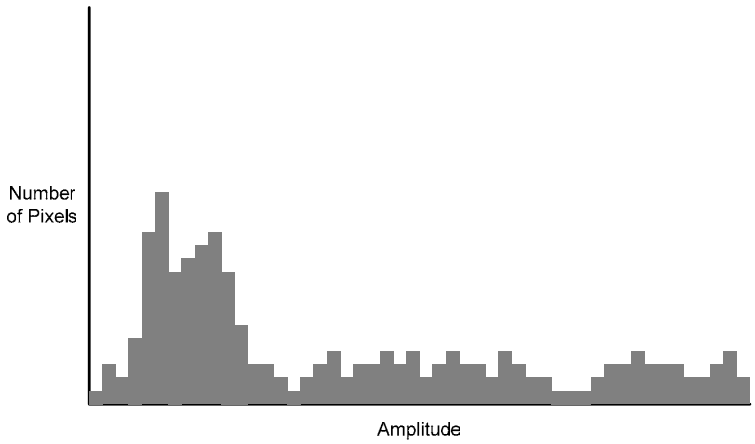

Figure 1: An example of an image histogram

If the generated numbers are uniformly distributed each count should be *N*/64. The program generates the (pseudo) random sequence with the code:

```
srand(1); // set rand() seed to 1 for repeatability

for(i=0;i<N;i++) {  // load array with digits
      a[i] = rand() % 64;  // Specify the number to be 0-63
}
```

The code then proceeds to compute the histogram. The histogram is stored in an array, hist[64].

[Part A]

Write the C sequential version of the histogram code for the CPU to generate the histogram array from the input array.

[Part B] histogram_kernel

Modify the sequential version of the sequential C code to be a CUDA program with GPU threads collectively performing the histogram calculation. The program is to use a 1-dimensional grid and a 1-dimensional block. Similar to the vector reduction, your resulting code will allocate a sub-histogram for each block. Suggested implementation are presented in lecture.

The requirements of your program are:

(1) Dynamically allocated memory for the holding the random sequence and keyboard input statements to be to specify the number of numbers, *N*.
(2) Add host code to compute the histogram on the host only.
(3) Add code to ensure both CPU and GPU versions of the calculation produce the same correct results
(4) Add program code statements to allow the user input different values for the size of the array, the number of histogram bins (data range) and the CUDA grid/block structure:

- Numbers of numbers (*N*)
- Numbers of threads in a block (*T*)
- Number of blocks in a grid (*B*)

Histogram values (H) fix the number of bins to 64 (no change)

(5) Display the final histogram numerically.
(6) Timing -- Add statements to time the execution of the code using CUDA events, both for the host-only (CPU) computation and with the device (GPU) computation, and display results. Compute the speed-up factor and display.

Your version of this code will use the GPU to compute partial histograms, and then use the CPU to tally up all of the histograms. If there were N=128, H=64, T=32, then B=4, and the GPU would generate 4 separate partial histograms of 64 integer entries. The CPU could add those partial histograms into one composite histogram (just like vector reduction).

Experiment with different input values (with at least six different values of *N* up to 1000000) and collect results on the Linux GPU server.

[Part C] histogram_atomic_kernel

You are to experiment with atomic support of GPU hardware, thus eliminating the need to sum up the partial histograms, as each block can atomically modify a single global histogram.

```
atomicAdd(d_Histogram[value], sum);
```

[Handin – Upload to the electronic system]

You must upload the following to moodle. Your results will be given points based on clarity and following what is asked.

[Item 1] - For the assignment, submit the .cu file and the results of the following. You will build a CUDA histogram solution without atomics.

Fill in the execution times using the event timers (T=32, H=64)

Array Size (N)	CPU Execution	GPU Execution + CPU clean up : **histogram_kernel**	GPU Speedup
1000			
10000			
100000			
1000000			

You should also include some graphs/analysis of the GPU memory overhead versus GPU computation.

[Item 2] – For the assignment, you need to insert the clock() function within the CUDA kernel to perform performance evaluation of the following:

Build a histogram of the block execution times (start of a block to completion of a block). Note- the goal is to classify the execution times of the blocks. Use T=32, H=64, N=10000

(a) Draw a histogram using Matlab or Microsoft Excel showing the block execution times.
(b) Draw a timeline showing when the blocks execute (start to finish)
(c) Answer the following questions:

What time execution for the longest running block?
What time execution for the shortest running block?
How much execution time occurs at the end of the execution (when all N-1 blocks have finished, but there is only 1 remaining block waiting to finish)?

[Item 3] – histogram_atomic_kernel: You are to examine an optimization through the use of an atomic instruction to enforce one thread at a time accessing to individual locations in the histogram array.

Array Size (N)	GPU Execution + CPU clean up : **histogram_kernel**	GPU Execution + CPU clean up : **histogram_atomic_kernel**	Speedup
1000			
10000			
100000			
1000000			

[Item 4] – You are to use the clock() to time the activity of "`atomicAdd`" You will need to build an output result array that each block of threads will write to the output time.

```
clock_t start_atomic;
clock_t stop_atomic;

start_atomic = clock();
atomicAdd(d_Histogram[value], s_Histogram[value]);
stop_atomic = clock();

if (tid == 0)
    atomic_timer[bid] = stop_atomic – start_atomic;
```

You will describe (Microsoft excel graphs, etc) the overhead of atomics within the GPU system.