Assignement-3 HPC(CO332)

By: GroupName :student19

Aswanth P P 15CO112

Arvind Ramachandran 15CO111

Question-1

1. Since each thread access global memory and copies values to its local and update it there, then copy back to global memory. So if there is a two thread try to access to same location each will overwrite the will written by other thread. Hence we cannot get the exact histogram distribution of the input.

2. What we did in getting the correct answer use of atomicAdd function to accumulate value for each thread. This will get the correct answer but each thread to access to same location will be in queue hence create a reduce the performance
3. The kernel is given input data array, the length of the input array, and a pointer to the output histogram. The kernel computes a liberalized offset into the input data array. Each thread will start with an offset between 0 and the number of threads minus 1. It will then stride by the total number of threads that have been launched.
4. Each thread reads one value from input array hence number of global reads equal to input length.
5. Each thread writes one value from input array hence number of global writes equal to number of elements in NUM_BINS that is 4096
6. Since atomic operation is done for each location in number of atomicAdd is equal to number of elements in NUM_BINS that is 4096
7. If each value in input array is same then we can use shared memory that each block gets a copy of array of size num_bins it updates the values locally. Then reduce it to global memory.
8. Another way we can use is sort the input array and take upper bound array index of a particular value. Then find frequency of each value by using adjacent difference.

Question-2

1. The problem with computing a histogram from the input data arises from the fact that multiple threads may want to increment the same bin of the output histogram at the same time. In this situation, we will need to use atomic increments to avoid this situation. When thousands of threads are trying to access a handful of memory locations, a great deal of contention for our histogram bins can occur. To ensure atomicity of the increment operations, the hardware needs to serialize operations to

the same memory location. This can result in a long queue of pending operations, and any performance gain we might have had will vanish.

2. Optimal performance is achieved when the number of blocks we launch is exactly twice the number of multiprocessors our GPU contains. For example, a GeForce GTX 280 has 30 multiprocessors, so our histogram kernel happens to run fastest on a GeForce GTX 280 when launched with 60 parallel blocks
3. Number of blocks created for optimised performance = 2 * Number of Multiprocessors in
   GPU = 2 *13 =26
   Total number of threads created = Threads per block * Number of blocks
   = 256 * 26
   = 6656
   Number of global reads per thread = 3
   Number of global reads = 2 * 6656 =13312
4. Number of blocks created for optimised performance = 2 * Number of Multiprocessors in
   GPU

   = 2 *13 =26

   Total number of threads created = Threads per block * Number of blocks
   = 256 * 26
   = 6656
   Number of global writes per thread = 1
   Number of global writes = 1 * 6656 =6656
5. Since we have decided to use 256 threads and have 256 histogram bins, each thread atomically adds a single bin to the final histogram's total. So number of atomic operations=no of threads created= 256.
6. In this case numbers of bins and threads in a block does not match. So this phase would be more complicated. We have no guarantees about what order the blocks add their values to the final histogram, but since integer addition is commutative, we will always get the same answer provided that the additions occur atomically. Hence number of atomic operations = Number of threads

Question-3

1. No, there is no implicit memory copy between the host and device
2. O(n+k) where n is Number of iterations for Input array initialization and k is Number of iterations for Count array computation.
3. **(n*logn)**

4. A straightforward implementation is subject to severe performance hazards. The processors based on the G80architecture (i.e., Compute Capability 1.0 and 1.1) impose strict conditions on which memory access patterns may benefit from memory coalescing.

Question-4

1. Blur filters, Edge detection algorithms and fundamental concepts in signal processing and analysis
2. 3*N FLOPS
   N Add
   2*N Mul
3. 2*N
4. N
5. min = TILE_WIDTH * TILE_WIDTH.
   max = image_height*image_width.
   Avg = MASK_HEIGHT*MASK_WIDTH
6. 8.18 seconds
7. 3*MASK_HEIGHT*MASK_WIDTH
8. Yes, If not used, too many conditional statements are needed and too much in place calculation will lead to slow running code which we don't want.
9. 64x64 matrix which is passed with keyword (__restrict). [to the kernel!]

Question-5

1. 12*TILE_WIDTH*NoOfBlocks
2. BlockSize*6*NoOfBlocks
3. More memory accesses.
   Because the answer is directly proportional to the number of blocks and now the number of blocks will increase for the same sized input in case of 3*3 as compared to 5*5.