EE147
Lab 3 Report: Histogram

Victoria Hall, 861154075

May 9, 2018

Questions

1. Use visual profiler to report relevant statistics (e.g. utilization and memory hierarchy related) about the execution of your kernels. Did you find any surprising results?

   Stats from the visual profiler based on the default case of 1,000,000 elements and 4096 bins:

   (a) Regs: 10
   (b) Shared Stored Transactions: 610,627
   (c) Shared Load Transactions: 610,627
   (d) Shared Memory Efficiency: 81.9%
   (e) Atomic transactions: 2,000,384
   (f) Warp Execution Efficiency: 100%
   (g) Warp Non-Predicated Execution Efficiency: 98.5%
   (h) Global Memory Load Efficiency: 100%
   (i) Global Memory Store Efficiency: 0%
   (j) Global Load Transactions: 250,000
   (k) Global Store Transactions: 0

   I was surprised that there were no global store transactions even though we do add the private histogram bins back to the main bins variable. This could be due to the way the atomicAdd function works though. It was also surprising that there were only 250,000 global load transactions which is only $\frac{1}{4}$ the number of elements in the entire array. I was also surprised that there were only 610,627 shared stored/load transactions since I would think there would be as many store/load transactions as there were elements in the array, but this is about 400,000 transactions short of the number of elements in the default case. The atomic transactions is close to twice the number of elements in the array, but that is expected since we use atomic add for the private histogram on all the input array and then use atomic add to add the private histogram to the final bins variable.

2. What, if any, limitations are there on m and n for your implementation? Explain these limitations and how you may overcome them with a different implementation.

   For m is the parameter for the number of elements in the array and n is the parameter for the number of bins. One limitation of the program is that if m is greater than 0 but n is 0, then there is an error reported: "Floating point exception (core dumped)." This could be improved by printing that an invalid number of bins was provided and allowing the user to select a new number of bins. The program works improperly when a negative number is input for m or n. In this case, the negative number creates an underflow problem so that the resulting number of elements or bins ends up being

$4,294,967,295 + (-n) + 1$ which is also the two's compliment of the number to make it negative in binary. Since the variables are unsigned ints, they don't handle this case and the number of elements or bins ends up being too large to allocate the memory on the device so the program results in an "Unable to allocate device memory" error. This could be fixed similarly to the first problem which is including more boundary conditions to check for when reading the input arguments of the system. Alternatively, the code could be changed to use ints that allow signs instead of declaring the variables as unsigned ints.