EE217 FINAL PROJECT

Option 1 Mordern CUDA

Histogram Implementation

By

Varshini Sampath

862251212 vsamp003@ucr.edu

https://github.com/UCR-CSEE217/final-project-vsampgpufinals

OVERVIEW

This project provides an implementation of the histogram algorithm using modern CUDA techniques like Unified Memory and CUDA Streams. Histogram is an algorithm where the input is an array and output is a set of bins and the algorithm decides which bin each of the element in the input must go into based on certain logic. This report talks about the implementation and compares it with the basic histogram algorithm without any modern techniques used.

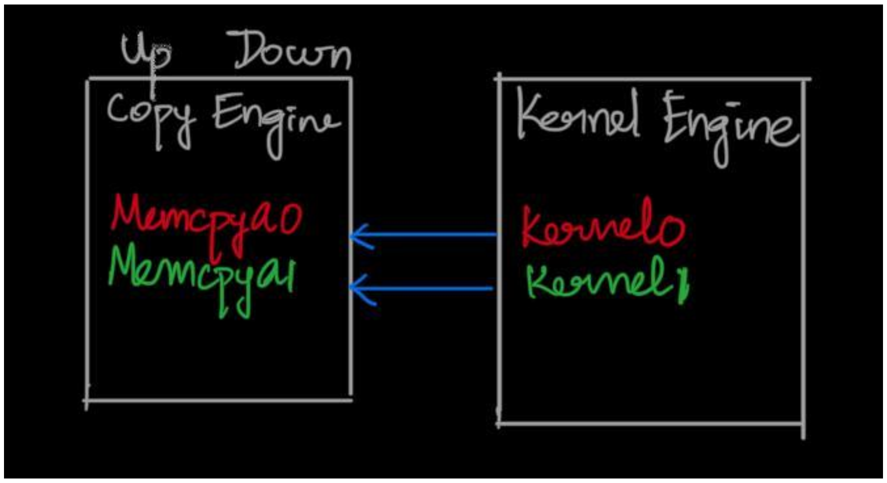
TECHNICAL DESCRIPTION

Optimizations used – Unified Memory, CUDA Streams

Detailed Explanation

Generally, malloc allocates in the main memory and when cudaMemCpy is called two transfers take place, from main memory to pinned memory (to ensure that the mapping from virtual address to physical address is persistent throughout the course of cudaMemCpy) and from pinned memory to the global memory (GPU). Here, **CudaHostAlloc** has been used to allocate the space for input directly in the pinned memory so that cudaMemCpy is faster. CudaMallocManaged has been used to enable simplified programming of data movement.

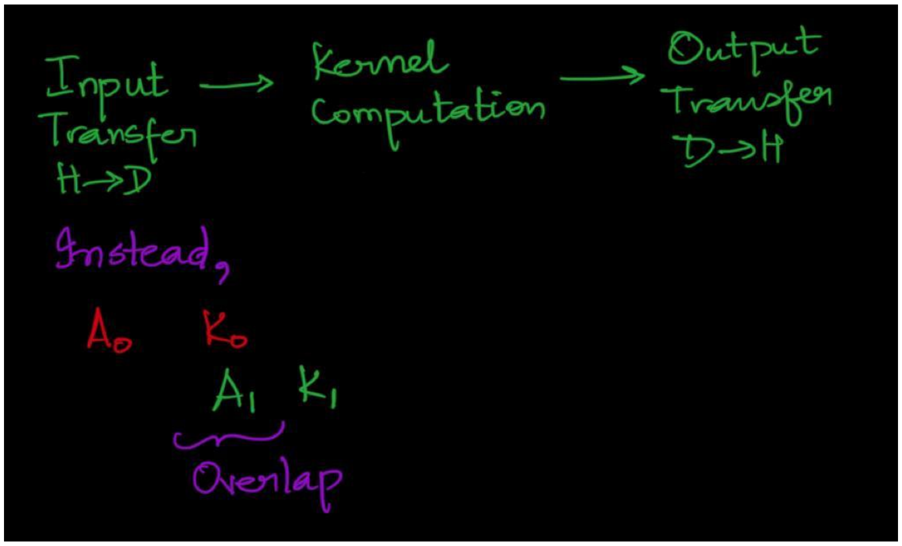
The main optimization is done using streams. We have used two streams. We have an input array and output bins array. Shifting the whole of the input array from host to device and then starting with the kernel computation takes lot of time. Instead here, we shift the input part by part and for the part shifted we start the computation while transferring the next part.



Each stream is a queue of operations. We push the tasks into the queue in the order we would want it to execute. Instead of transferring the input entirely from host to device and then proceeding with computation, this kind of splitting the tasks cause overlapping operations, thus using up and down link of PCIe and kernel engine parallelly and more efficiently. Thus making the algorithm faster.

While pushing the MemCpy into the queue, we use cudaMemcpyAsync. The general cudaMemcpy is blocking and it wont let us move to the next line until it finishes. But cudaMemcpyAsync is non-blocking. We can populate things in the queue and continue.

In the current problem, we have used two streams and tried to overlap the memcpy host to device and kernel computation as shown below.



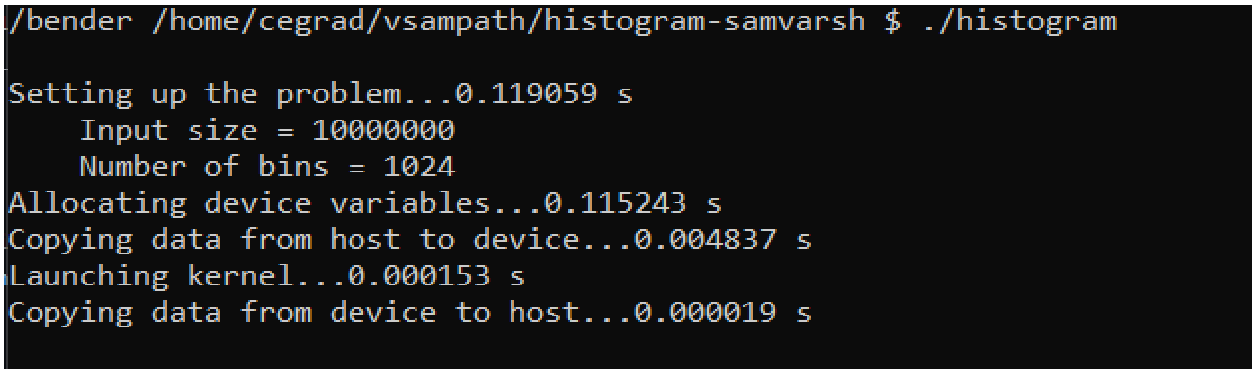
STATUS OF THE PROJECT

Continuing with the above logic, each time the bins array that is sent is of full size. The bins is not split up like the input. The number of bins can't be more than 4096. The atomic addition keeps happening in the same two bins. Finally, a vecadd kernel is used to add them up, the output of which is transferred to the host. This requires only one device to host transfer and that is why it is not added to the streams. The segment size for each of the stream is by default set to 100000 to efficiently use the threads in each of the kernel computation (histogram). But a small logic is added to update it based on the number of input elements. Also the code handles only even number of input elements.

The feature is complete and the algorithm works.

EVALUATION AND RESULTS

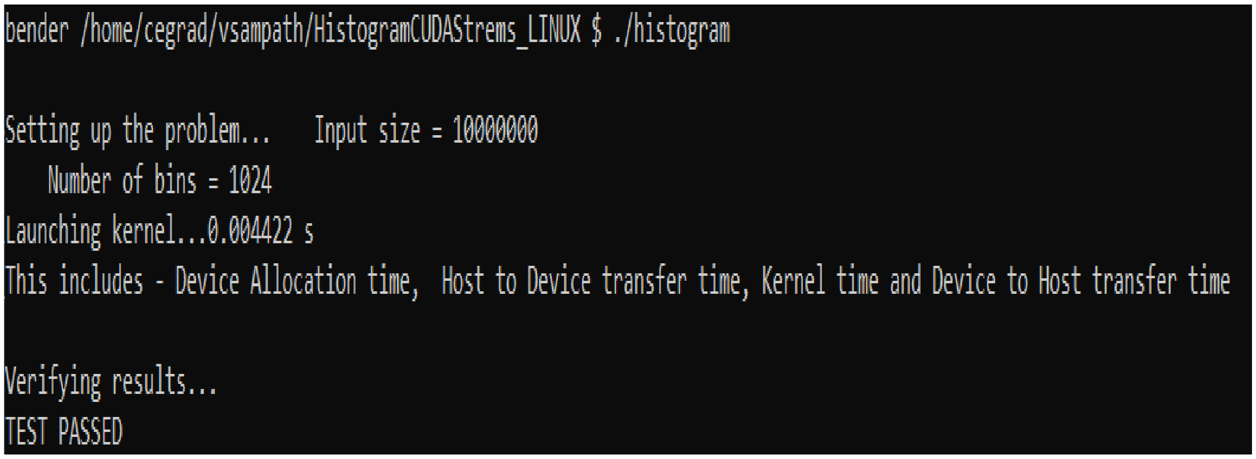
The basic histogram kernel with privatization technique gives the following output. We can see that the total time taken for allocating device variables, data transfer and kernel computation is **1.120252 s**.



```
/bender /home/cegrad/vsampath/histogram-samvarsh $ ./histogram

Setting up the problem...0.119059 s
    Input size = 10000000
    Number of bins = 1024
Allocating device variables...0.115243 s
Copying data from host to device...0.004837 s
Launching kernel...0.000153 s
Copying data from device to host...0.000019 s
```

The histogram algorithm with streams and unified memory gives the following output. We can see that the total time taken here is **0.004422 s.** This is inclusive of the device allocation time, transfer time and the kernel computation time.



```
bender /home/cegrad/vsampath/HistogramCUDAStrems_LINUX $ ./histogram

Setting up the problem...    Input size = 10000000
    Number of bins = 1024
Launching kernel...0.004422 s
This includes - Device Allocation time,  Host to Device transfer time, Kernel time and Device to Host transfer time

Verifying results...
TEST PASSED
```

Comparing both the implementations, we see that the one with streams give a significantly better performance.

Now, lets look at the time for various inputs

INPUT, BINS	Without Streams (in sec)	With Streams (in sec)
100, 10	0.108467	0.008666
1000, 10	0.167301	0.003532
100000, 100	0.105326	0.000614
1M, 1024	0.302888	0.001550
100M, 1024	0.417481	0.037571

We see that streams perform better.

HOW TO COMPILE AND RUN

Similar to the assignments. Compile using make. Run using command ./histogram.

The algorithm gives a time, which is the time taken for allocation of device variables, transferring data from host to device, kernel computations and transferring data back from device to host.