GPU Project 3: Parallel Radix Partition

Course: CIS6930/4930 Massive Parallel Computing
Instructor: Dr. Yi-Cheng Tu

In this project, our task was to use CUDA to implement a GPU version of the Parallel Radix Partition and achieve an optimized result by exploiting the GPU. Apart from the code, which was already provided, the program is mainly divided into three parts which were implemented by me and these are:

1. Histogram
   Input: the input array of keys, the target number of partitions.
   Output: the histogram of the partitions.

Algorithm 1 histogram

```
for each k in keys do
  h = bit_extract(k,number_of_bits);
  histogram[h]++;
end for
return histogram;
```

2. Prefix Scan
   Input: the histogram of the partitions.
   Output: the exclusive prefix sum of the histogram.

Algorithm 2 prefix_scan

```
for i from 0 to number_of_partitions do
  if i == 0 then
    sum[i] = 0;
  else
    sum[i] = histogram[i-1]+sum[i-1];
  end if
end for
return sum;
```

3. Reorder
   Input: the array of keys, the prefix sum of the histogram.
   Output: the reordered array of keys.

Algorithm 3 reorder

```
for each k in keys do
  h = bit_extract(k,number_of_bits);
  offset = atomicAdd(&sum[h],1);
  output[offset] = k;
end for
```

One of the best uses of GPU programming is parallelizing a loop using a kernel, especially using stride to parallelize the loop. So, in stride implementation, the product of blockDim.x and gridDim.x provides the cumulative number of grid threads. For instance, if the grid includes 512 threads, thread 0 will compute elements 0, 512, 1024, etc. So, we have parallelized all the loops from the histogram, prefix scan and reorder. We can ensure that all addressing within warps is unit-stride by using a loop with a stride equal to the grid size, so we get full memory coalescing, I implemented this technique to calculate the histogram.

Output:

```
(base) ghalibsaleem@Ghalib-linux:~/Documents/USF/Cuda/poj3-ghalibsaleem$ ./proj3 1000 8
Partition 0:  Offset: 0  Number of Keys: 125
Partition 1:  Offset: 125  Number of Keys: 125
Partition 2:  Offset: 250  Number of Keys: 125
Partition 3:  Offset: 375  Number of Keys: 125
Partition 4:  Offset: 500  Number of Keys: 125
Partition 5:  Offset: 625  Number of Keys: 125
Partition 6:  Offset: 750  Number of Keys: 125
Partition 7:  Offset: 875  Number of Keys: 125

Running Time for all kernals: 0.000189s
(base) ghalibsaleem@Ghalib-linux:~/Documents/USF/Cuda/poj3-ghalibsaleem$
(base) ghalibsaleem@Ghalib-linux:~/Documents/USF/Cuda/poj3-ghalibsaleem$
(base) ghalibsaleem@Ghalib-linux:~/Documents/USF/Cuda/poj3-ghalibsaleem$ ./proj3 65000 16
Partition 0:  Offset: 0  Number of Keys: 4063
Partition 1:  Offset: 4063  Number of Keys: 4063
Partition 2:  Offset: 8126  Number of Keys: 4063
Partition 3:  Offset: 12189  Number of Keys: 4063
Partition 4:  Offset: 16252  Number of Keys: 4063
Partition 5:  Offset: 20315  Number of Keys: 4063
Partition 6:  Offset: 24378  Number of Keys: 4063
Partition 7:  Offset: 28441  Number of Keys: 4063
Partition 8:  Offset: 32504  Number of Keys: 4062
Partition 9:  Offset: 36566  Number of Keys: 4062
Partition 10:  Offset: 40628  Number of Keys: 4062
Partition 11:  Offset: 44690  Number of Keys: 4062
Partition 12:  Offset: 48752  Number of Keys: 4062
Partition 13:  Offset: 52814  Number of Keys: 4062
Partition 14:  Offset: 56876  Number of Keys: 4062
Partition 15:  Offset: 60938  Number of Keys: 4062

Running Time for all kernals: 0.007844s
(base) ghalibsaleem@Ghalib-linux:~/Documents/USF/Cuda/poj3-ghalibsaleem$
```