Counting Sort

Prof. Naga Kandasamy
ECE Department, Drexel University

The problem, worth 15 points, is due March 16, 2021. You may work on this problem in a team of up to two people. One submission per group will suffice. Please submit original work.

Counting sort is an efficient non-comparison based algorithm to sort an array of integers in the range 0 to r for some integer r. Given an input array comprising n elements, counting sort runs in $O(n)$ time, and trades the time complexity for space complexity when compared to other sorting algorithms. Let us walk through an example of how this algorithm operates. Consider the following input array with ten randomly generated integers, each in the range between 0 and 10:

```
input_array = 8 5 1 3 7 8 6 5 3 8
```

The first step is to generate a histogram with $10 + 1 = 11$ bins — one for each of the integers that could possibly appear in the input array. This histogram simply counts the number of times an integer occurs in the input array.

```
int bin[11]; /* Bins in the histogram. */
/* Count occurrence frequency of each input element. */
for (int i = 0; i < 10; i++)
    bin[input_array[i]]++;
```

For our input array, the resulting histogram has the following entries:

```
bin[0] bin[1] bin[2] bin[3] bin[4] bin[5] bin[6] bin[7] bin[8]
  0      1      0      2      0      2      1      1      3
bin[9] bin[10]
  0      0
```

We then perform an *inclusive prefix scan* of the above bin values.

```
int num_bins = 11;

for (i = 1; i < num_bins; i++)
        bin[i] = bin[i - 1] + bin[i];
```

The result of the prefix scan is as follows:

```
bin[0] bin[1] bin[2] bin[3] bin[4] bin[5] bin[6] bin[7] bin[8]
  0      1      1      3      3      5      6      7      10

bin[9] bin[10]
 10      10
```

The scanned value associated with each bin serves as an index into the output array to place the sorted elements. The following code snippet shows this process.

```
int start_idx = 0;
int i, j;
for (i = 0; i < num_bins; i++){
    for (j = start_idx; j < bin[i]; j++){
        sorted_array[j] = i;
    }
    start_idx = bin[i];
}
```

Since `bin[0] = 0`, nothing is written to the output array; `bin[1] = 1` and so the corresponding bin number is written to `sorted_array[0]`; the bin number associated with `bin[3]` is written twice—to output locations `sorted_array[1]` and `sorted_array[2]`; and so on. At the end of this process, we obtain the sorted array

```
sorted_array = 1 3 3 5 5 6 7 8 8 8
```

The code provided to you takes the number of elements to be sorted as a command-line parameter. Edit the *compute_on_device()* and *counting_sort_kernel()* functions to complete counting sort on the GPU. For full credit, your code must implement all three major steps on the GPU: histogram generation (**5 points**), inclusive scan (**5 points**), and generating the sorted array (**5 points**). You may add additional kernels and host-side code as needed.

Submit the files needed to build and run your code as a single zip file via BBLearn. Also, provide a short report describing how you designed your GPU code, using code or pseudocode if that helps the discussion, and the amount of speedup obtained over the serial version for 10^6 and 10^8 integers in the input array. Do not change the default range of 255 specified in the code. Include the overhead due to CPU/GPU communication when reporting the performance.