Module 7 Lab

Thrust Histogram Sort

GPU Teaching Kit – Accelerated Computing

OBJECTIVE

The purpose of this lab is to implement a histogramming algorithm for an input array of integers. This approach composes several distinct algorithmic steps to compute a histogram, which makes Thrust a valuable tools for its implementation.

PROBLEM SETUP

Consider the dataset

```
input = [2 1 0 0 2 2 1 1 1 1 4]
```

A resulting histogram would be

```
histogram = [2 5 3 0 1]
```

reflecting 2 zeros, 5 ones, 3 twos, 0 threes, and one 4 in the input dataset.

Note that the number of bins is equal to

```
max(input) + 1
```

HISTOGRAM SORT APPROACH

First, sort the input data using `thrust::sort`. Continuing with the original example:

```
sorted = [0 0 1 1 1 1 1 2 2 2 4]
```

Determine the number of bins by inspecting the last element of the list and adding 1:

```
num_bins = sorted.back() + 1
```

To compute the histogram, we can compute the culumative histogram and then work backwards. To do this in Thrust, use `thrust::upper_bound`. `upper_bound` takes an input data range (the sorted input) and a set of search values, and for each search value will report the largest index in the input range that the search value could be inserted into without changing the sorted order of the inputs. For example,

```
[2 8 11 11 12] = thrust::upper_bound([0 0 1 1 1 1 1 2 2 2 4], // input
                                     [0 1 2 3 4])             // search
```

By carefully crafting the search data, `thrust::upper_bound` will produce a cumulative histogram. The search data must be a range `[0,num_bins)`.

Once the cumulative histogram is produced, use `thrust::adjacent_different` to compute the histogram.

```
[2 5 3 0 1] = thrust::adjacent_difference([2 8 11 11 12])
```

Check the thrust documentation for details of how to use `upper_bound` and `adjacent_difference`. Instead of constructing the search array in device memory, you may be able to use `thrust::counting_iterator`.

PREREQUISITES

Before starting this lab, make sure that you have completed all of the Module 7 lecture videos.

INSTRUCTIONS

Edit the code in the code tab to perform the following:

- allocate space for input on the GPU
- copy host memory to device
- invoke thrust functions
- copy results from device to host

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

LOCAL SETUP INSTRUCTIONS

The most recent version of source code for this lab along with the build-scripts can be found on the Bitbucket repository. A description on how to use the CMake tool in along with how to build the labs for local development found in the README document in the root of the repository.

The executable generated as a result of compiling the lab can be run using the following command:

```
./ThrustHistogramSort_Template -e <expected.raw> \
  -i <input.raw> -o <output.raw> -t integral_vector
```

where `<expected.raw>` is the expected output, `<input.raw>` is the input dataset, and `<output.raw>` is an optional path to store the results. The datasets can be generated using the dataset generator built as part of the compilation process.

ATTRIBUTION

This is a simplified version of the material presented in the Thrust repository here.

QUESTIONS

(1) Are there places in your solution where there is an implicit memory copy between the host and device (a copy that is not from `thrust::copy`)?

ANSWER: **Probably: for example, a line like `num_bins = device_vector.back() + 1` copies the last element of a device vector back to the host implicitly.**

(2) What is the asympotic runtime of this approach to histogram ("bigO" time)?

ANSWER: **O(n log(n))**

(3) What is the asympotic runtime of the privatized atomic approach to histogram?

ANSWER: **O(n)**

(4) Why might the developers of Thrust not provide a straightforward `thrust::histogram()` interface?

ANSWER: **The performance of histogram can strongly depend on whether the histogram is stored sparsely or densely, whether or not values are histogrammed spearately than keys, the specific implementation of atomics on a GPU, etc. It is difficult to provide a catch-all fast histogram implementation.**

CODE TEMPLATE

The following code is suggested as a starting point for students. The code handles the import and export as well as the checking of the solution. Students are expected to insert their code is the sections demarcated with `//@@`. Students expected the other code unchanged. The tutorial page describes the functionality of the `wb*` methods.

```
1 #include <wb.h>
2
3 #include <thrust/adjacent_difference.h>
4 #include <thrust/binary_search.h>
```

```
#include <thrust/copy.h>
#include <thrust/device_vector.h>
#include <thrust/iterator/counting_iterator.h>
#include <thrust/sort.h>

int main(int argc, char *argv[]) {
  wbArg_t args;
  int inputLength, num_bins;
  unsigned int *hostInput, *hostBins;

  args = wbArg_read(argc, argv);

  wbTime_start(Generic, "Importing data and creating memory on host");
  hostInput = (unsigned int *)wbImport(wbArg_getInputFile(args, 0),
                                       &inputLength, "Integer");
  wbTime_stop(Generic, "Importing data and creating memory on host");

  wbLog(TRACE, "The input length is ", inputLength);

  // Copy the input to the GPU
  wbTime_start(GPU, "Allocating GPU memory");
  //@@ Insert code here
  wbTime_stop(GPU, "Allocating GPU memory");

  // Determine the number of bins (num_bins) and create space on the host
  //@@ insert code here
  num_bins = deviceInput.back() + 1;
  hostBins = (unsigned int *)malloc(num_bins * sizeof(unsigned int));

  // Allocate a device vector for the appropriate number of bins
  //@@ insert code here

  // Create a cumulative histogram. Use thrust::counting_iterator and
  // thrust::upper_bound
  //@@ Insert code here

  // Use thrust::adjacent_difference to turn the culumative histogram
  // into a histogram.
  //@@ insert code here.

  // Copy the histogram to the host
  //@@ insert code here

  // Check the solution is correct
  wbSolution(args, hostBins, num_bins);

  // Free space on the host
  //@@ insert code here
  free(hostBins);

  return 0;
}
```

CODE SOLUTION

The following is a possible implementation of the lab. This solution is intended for use only by the teaching staff and should not be distributed to students.

```
#include <wb.h>

#include <thrust/adjacent_difference.h>
#include <thrust/binary_search.h>
#include <thrust/copy.h>
#include <thrust/device_vector.h>
#include <thrust/iterator/counting_iterator.h>
#include <thrust/sort.h>

int main(int argc, char *argv[]) {
  wbArg_t args;
  int inputLength, num_bins;
  unsigned int *hostInput, *hostBins;

  args = wbArg_read(argc, argv);

  wbTime_start(Generic, "Importing data and creating memory on host");
  hostInput = (unsigned int *)wbImport(wbArg_getInputFile(args, 0),
                                       &inputLength, "Integer");
  wbTime_stop(Generic, "Importing data and creating memory on host");

  wbLog(TRACE, "The input length is ", inputLength);

  // Copy the input to the GPU
  wbTime_start(GPU, "Allocating GPU memory");
  //@@ Insert code here
  thrust::device_vector<int> deviceInput(inputLength);
  thrust::copy(hostInput, hostInput + inputLength, deviceInput.begin());
  wbTime_stop(GPU, "Allocating GPU memory");

  // Determine the number of bins (num_bins) and create space on the host
  //@@ insert code here
  thrust::sort(deviceInput.begin(), deviceInput.end());
  num_bins = deviceInput.back() + 1;
  hostBins = (unsigned int *)malloc(num_bins * sizeof(unsigned int));

  // Allocate a device vector for the appropriate number of bins
  //@@ insert code here
  thrust::device_vector<int> deviceBins(num_bins);

  // Create a cumulative histogram. Use thrust::counting_iterator and
  // thrust::upper_bound
  //@@ Insert code here
  thrust::counting_iterator<int> search_begin(0);
  thrust::upper_bound(deviceInput.begin(), deviceInput.end(), search_begin,
                      search_begin + num_bins, deviceBins.begin());

```

```
48    // Use thrust::adjacent_difference to turn the culumative histogram
49    // into a histogram.
50    //@@ insert code here.
51    thrust::adjacent_difference(deviceBins.begin(), deviceBins.end(),
52                                deviceBins.begin());
53
54    // Copy the histogram to the host
55    //@@ insert code here
56    thrust::copy(deviceBins.begin(), deviceBins.end(), hostBins);
57
58    // Check the solution is correct
59    wbSolution(args, hostBins, num_bins);
60
61    // Free space on the host
62    //@@ insert code here
63    free(hostBins);
64
65    return 0;
66  }
```