Module 3 Lab

CUDA Thrust Vector Add

GPU Teaching Kit – Accelerated Computing

OBJECTIVE

The purpose of this lab is to introduce the student to the CUDA API by implementing vector addition using Thrust.

PREREQUISITES

Before starting this lab, make sure that:

- You have completed all of Module 2 in the teaching kit
- You have completed the "Device Query" lab

INSTRUCTIONS

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

- Generate a `thrust::dev_ptr<float>` for host input arrays
- Copy host memory to device
- Invoke `thrust::transform()`
- 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:

```
./ThrustVectorAdd_Template -e <expected.raw> \
  -i <input0.raw>,<input1.raw> -o <output.raw> -t vector
```

where `<expected.raw>` is the expected output, `<input0.raw>,<input1.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.

QUESTIONS

(1) How many floating operations are being performed in your vector add kernel? EXPLAIN.

ANSWER: **N - one for each pair of input vector elements.**

(2) How many global memory reads are being performed by your kernel? EXPLAIN.

ANSWER: **2N - one for each input vector element.**

(3) How many global memory writes are being performed by your kernel? EXPLAIN.

ANSWER: **N - one for each output vector element.**

(4) In what ways did Thrust make developing a functional vector addition code easier or harder?

ANSWER: **Declaring device data as vectors reduces the amount of host code. The actual code that invokes the vector addition is a more functional style than the imperative CUDA kernel.**

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.

```
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>
#include <wb.h>

int main(int argc, char *argv[]) {
  wbArg_t args;
  float *hostInput1 = nullptr;
  float *hostInput2 = nullptr;
  float *hostOutput = nullptr;
  int inputLength;

  args = wbArg_read(argc, argv); /* parse the input arguments */
```

```
13
14    // Import host input data
15    wbTime_start(Generic, "Importing data to host");
16    hostInput1 =
17        (float *)wbImport(wbArg_getInputFile(args, 0), &inputLength);
18    hostInput2 =
19        (float *)wbImport(wbArg_getInputFile(args, 1), &inputLength);
20    wbTime_stop(Generic, "Importing data to host");
21
22    // Declare and allocate host output
23    //@@ Insert code here
24    wbTime_start(GPU, "Doing GPU Computation (memory + compute)");
25
26    // Declare and allocate thrust device input and output vectors
27    wbTime_start(GPU, "Doing GPU memory allocation");
28    //@@ Insert code here
29    wbTime_stop(GPU, "Doing GPU memory allocation");
30
31    // Copy to device
32    wbTime_start(Copy, "Copying data to the GPU");
33    //@@ Insert code here
34    wbTime_stop(Copy, "Copying data to the GPU");
35
36    // Execute vector addition
37    wbTime_start(Compute, "Doing the computation on the GPU");
38    //@@ Insert Code here
39    wbTime_stop(Compute, "Doing the computation on the GPU");
40    ////////////////////////////////////////////////////////
41
42    // Copy data back to host
43    wbTime_start(Copy, "Copying data from the GPU");
44    //@@ Insert code here
45    wbTime_stop(Copy, "Copying data from the GPU");
46
47    wbTime_stop(GPU, "Doing GPU Computation (memory + compute)");
48
49    wbSolution(args, hostOutput, inputLength);
50
51    free(hostInput1);
52    free(hostInput2);
53    free(hostOutput);
54    return 0;
55  }
```

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.

```
1  #include <thrust/device_vector.h>
2  #include <thrust/host_vector.h>
```

```cpp
#include <wb.h>

int main(int argc, char *argv[]) {
  wbArg_t args;
  float *hostInput1;
  float *hostInput2;
  float *hostOutput;
  int inputLength;

  args = wbArg_read(argc, argv); /* parse the input arguments */

  // Import host input data
  wbTime_start(Generic, "Importing data to host");
  hostInput1 =
      (float *)wbImport(wbArg_getInputFile(args, 0), &inputLength);
  hostInput2 =
      (float *)wbImport(wbArg_getInputFile(args, 1), &inputLength);
  wbTime_stop(Generic, "Importing data to host");

  // Declare and allocate host output
  //@@ Insert code here
  hostOutput = (float *)malloc(sizeof(float) * inputLength);

  wbTime_start(GPU, "Doing GPU Computation (memory + compute)");

  // Declare and allocate thrust device input and output vectors
  wbTime_start(GPU, "Doing GPU memory allocation");
  //@@ Insert code here
  thrust::device_vector<float> deviceInput1(inputLength);
  thrust::device_vector<float> deviceInput2(inputLength);
  thrust::device_vector<float> deviceOutput(inputLength);
  wbTime_stop(GPU, "Doing GPU memory allocation");

  // Copy to device
  wbTime_start(Copy, "Copying data to the GPU");
  //@@ Insert code here
  thrust::copy(hostInput1, hostInput1 + inputLength, deviceInput1.begin());
  thrust::copy(hostInput2, hostInput2 + inputLength, deviceInput2.begin());
  wbTime_stop(Copy, "Copying data to the GPU");

  // Execute vector addition
  wbTime_start(Compute, "Doing the computation on the GPU");
  //@@ Insert Code here
  thrust::transform(deviceInput1.begin(), deviceInput1.end(),
                    deviceInput2.begin(), deviceOutput.begin(),
                    thrust::plus<float>());
  wbTime_stop(Compute, "Doing the computation on the GPU");
  ////////////////////////////////////////////////////////

  // Copy data back to host
  wbTime_start(Copy, "Copying data from the GPU");
  //@@ Insert code here
  thrust::copy(deviceOutput.begin(), deviceOutput.end(), hostOutput);
```

```
56    wbTime_stop(Copy, "Copying data from the GPU");
57
58    wbTime_stop(GPU, "Doing GPU Computation (memory + compute)");
59
60    wbSolution(args, hostOutput, inputLength);
61
62    free(hostInput1);
63    free(hostInput2);
64    free(hostOutput);
65    return 0;
66  }
```