Module 10 Lab

Thrust List Reduction

GPU Teaching Kit – Accelerated Computing

OBJECTIVE

Implement a kernel to perform an inclusive prefix scan on a 1D list using Thrust.

Given an input

```
x = [x0, x1, x2, ...]
```

Produce an output

```
y = [y0, y1, y2, ...]
```

where

```
y[0] = 0
y[1] = 0 + x[0]
y[2] = 0 + x[0] + x[1]
y[i] = y[i-1] + x[i-1]
```

The prefix scan should produce `y` given `x` and use the `thrust::inclusive_scan` function. The input and output will both be `float` arrays of equal length.

PREREQUISITES

Before starting this lab, make sure that:

- You have completed the required course modules
- You have completed the Thrust vector addition 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::inclusive_scan`

- 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:

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

QUESTIONS

(1) Name 3 applications of scan.

ANSWER: **Sorting, resource allocation, numerical integeration.**

(2) Suppose a you want to perform the algorithm using a binary operator that's not commutative, can you use still use parallel scan?

ANSWER: **To parallelize a scan the operator must be associative. Some operators, such as subtraction, are not associative but can be transformed into an associative operation. For subtraction, for example, we can transform it into a plus operation:** $a - b == a + (-b)$.

(3) Is it possible to get different results from running the serial version and parallel version of reduction? EXPLAIN.

ANSWER: **The order of floating-point operations in the serial and parallel version is different, so it is possible.**

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 *hostInput, *hostOutput; // The input 1D list
  int num_elements;              // number of elements in the input list

  args = wbArg_read(argc, argv);

  wbTime_start(Generic, "Importing data and creating memory on host");
  hostInput =
      (float *)wbImport(wbArg_getInputFile(args, 0), &num_elements);
  wbTime_stop(Generic, "Importing data and creating memory on host");

  wbLog(TRACE, "The number of input elements in the input is ",
        num_elements);

  // Declare and allocate the host output array
  //@@ Insert code here

  // Declare and allocate thrust device input and output vectors
  wbTime_start(GPU, "Allocating GPU memory.");
  //@@ Insert code here
  wbTime_stop(GPU, "Allocating GPU memory.");

  // Execute vector addition
  wbTime_start(
      Compute,
      "Doing the computation on the GPU and copying data back to host");
  //@@ Insert Code here
  wbTime_stop(Compute, "Doing the computation on the GPU");

  wbSolution(args, hostOutput, num_elements);

  // Free Host Memory
  free(hostInput);
  //@@ Insert code here

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

```
4
5  int main(int argc, char **argv) {
6    wbArg_t args;
7    float *hostInput, *hostOutput; // The input 1D list
8    int num_elements;              // number of elements in the input list
9
10   args = wbArg_read(argc, argv);
11
12   wbTime_start(Generic, "Importing data and creating memory on host");
13   hostInput =
14       (float *)wbImport(wbArg_getInputFile(args, 0), &num_elements);
15   wbTime_stop(Generic, "Importing data and creating memory on host");
16
17   wbLog(TRACE, "The number of input elements in the input is ",
18         num_elements);
19
20   // Declare and allocate the host output array
21   //@@ Insert code here
22   hostOutput = (float *)malloc(num_elements * sizeof(float));
23
24   // Declare and allocate thrust device input and output vectors
25   wbTime_start(GPU, "Allocating GPU memory.");
26   //@@ Insert code here
27   thrust::device_vector<float> deviceInput(num_elements);
28   thrust::device_vector<float> deviceOutput(num_elements);
29   thrust::copy(hostInput, hostInput + num_elements, deviceInput.begin());
30   wbTime_stop(GPU, "Allocating GPU memory.");
31
32   // Execute vector addition
33   wbTime_start(
34       Compute,
35       "Doing the computation on the GPU and copying data back to host");
36   //@@ Insert Code here
37   thrust::inclusive_scan(deviceInput.begin(), deviceInput.end(),
38                          deviceOutput.begin());
39   thrust::copy(deviceOutput.begin(), deviceOutput.end(), hostOutput);
40
41   wbTime_stop(Compute, "Doing the computation on the GPU");
42
43   wbSolution(args, hostOutput, num_elements);
44
45   free(hostInput);
46   free(hostOutput);
47
48   return 0;
49 }
```

This work is licensed by UIUC and NVIDIA (2016) under a Creative Commons Attribution-NonCommercial 4.0 License.