15-418/15-618 Parallel Computer Architecture and Programming

Homework 2, A Simple CUDA Renderer

Siqi Guo(AndrewID: siqiguo) January 16, 2025

1. CUDA Warm-Up 1: SAXPY (5 pts)

To gain a bit of practice writing CUDA programs, the warm-up task is to implement the SAXPY function.

i. `./saxpy`.

```
---------------------------------------------------------
Found 1 CUDA devices
Device 0: NVIDIA GeForce RTX 2080
   SMs:       46
   Global mem: 7959 MB
   CUDA Cap: 7.5
---------------------------------------------------------
Overall: 74.850 ms          [2.986 GB/s]
   1.006 ms            [222.288 GB/s]
   Total Bytes:240000000Overall: 27.106 ms        [8.246 GB/s]
   1.010 ms            [221.262 GB/s]
   Total Bytes:240000000Overall: 27.158 ms        [8.230 GB/s]
   1.008 ms            [221.678 GB/s]
```

ii. `./cudaSaxpy`.

```
---------------------------------------------------------
Found 1 CUDA devices
Device 0: NVIDIA GeForce RTX 2080
   SMs:       46
   Global mem: 7959 MB
   CUDA Cap: 7.5
---------------------------------------------------------
Kernel: 0.686 ms            [325.663 GB/s]
Overall: 24.608 ms          [9.083 GB/s]
Kernel: 0.685 ms            [326.524 GB/s]
Overall: 25.407 ms          [8.798 GB/s]
Kernel: 0.683 ms            [327.363 GB/s]
Overall: 25.375 ms          [8.808 GB/s]
```

2. CUDA Warm-Up 2: Parallel Prefix-Sum (10 pts)

Find peaks.

i. The implementation of `cudaScan()` (to get exclusive prefix sum) has been completed and passed the correctness test.

```
siqiguo@ghc28:~/private/15-618-Parallel-Computing/asst2/scan$ ./checker.pl -m scan
Mode: scan
Input: random

--------------
Running tests:
--------------

Element Count: 10000
Correctness passed!
Your Time: 0.173
Target Time: 0.184

Element Count: 100000
Correctness passed!
Your Time: 0.285
Target Time: 0.269

Element Count: 1000000
Correctness passed!
Your Time: 0.755
Target Time: 0.443

Element Count: 2000000
Correctness passed!
Your Time: 1.406
Target Time: 0.842

-------------------------
Scan Score Table:
-------------------------
-------------------------------------------------------------------------
| Element Count | Target Time  | Your Time     | Score         |
-------------------------------------------------------------------------
| 10000         | 0.184        | 0.173         | 1.25          |
| 100000        | 0.269        | 0.285         | 1.25          |
| 1000000       | 0.443        | 0.755         | 0.880132450331126 |
| 2000000       | 0.842        | 1.406         | 0.898293029871977 |
-------------------------------------------------------------------------
|                             | Total score:  | 4.2784254802031/5 |
-------------------------------------------------------------------------
```

ii. The implementation of `find_peaks()` (to get exclusive prefix sum) has been completed and passed the correctness test.

```
siqiguo@ghc28:~/private/15-618-Parallel-Computing/asst2/scan$ ./checker.pl -m find_peaks
Mode: find_peaks
Input: random

--------------
Running tests:
--------------

Element Count: 10000
Correctness passed!
Your Time: 0.203
Target Time: 0.209

Element Count: 100000
Correctness passed!
Your Time: 0.309
Target Time: 0.299

Element Count: 1000000
Correctness passed!
Your Time: 1.049
Target Time: 0.551

Element Count: 2000000
Correctness passed!
Your Time: 1.848
Target Time: 1.064

-------------------------
Find_peaks Score Table:
-------------------------
-------------------------------------------------------------------------
| Element Count | Target Time  | Your Time     | Score         |
-------------------------------------------------------------------------
| 10000         | 0.209        | 0.203         | 1.25          |
| 100000        | 0.299        | 0.309         | 1.25          |
| 1000000       | 0.551        | 1.049         | 0.78789323164919 |
| 2000000       | 1.064        | 1.848         | 0.863636363636364 |
-------------------------------------------------------------------------
|                           | Total score:  | 4.15152959528555/5 |
-------------------------------------------------------------------------
```

3. A Simple Circle Renderer (85 pts)

An implementation of a renderer that draws colored circles.

i. First, given starter code.

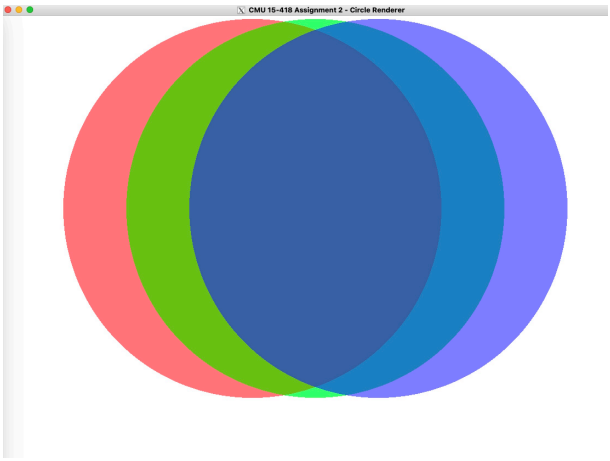

Figure 1: `./render rgb`

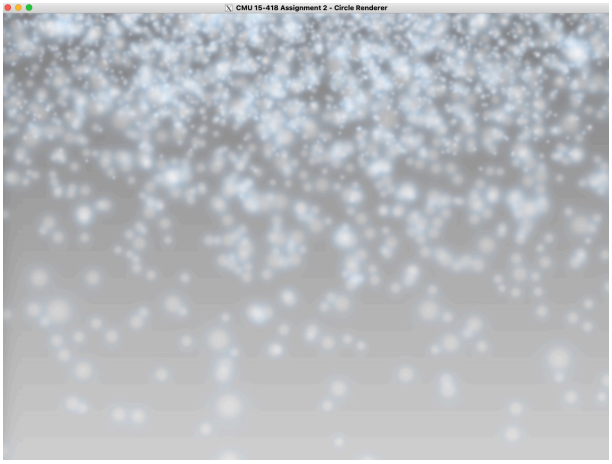

Figure 2: `./render snow`

Machine Info:

```
---------------------------------------------------------
Found 1 CUDA devices
Device 0: NVIDIA GeForce RTX 2080
   SMs:       46
   Global mem: 7959 MB
   CUDA Cap: 7.5
---------------------------------------------------------
```

Run on starter code.

```
siqiguo@ghc28:~/private/15-618-Parallel-Computing/asst2/render$ make check
mkdir -p objs/
g++ -m64 -O3 -Wall -g -o render objs/main.o objs/display.o objs/benchmark.o objs/refRenderer.o
    objs/cudaRenderer.o objs/noise.o objs/ppm.o objs/sceneLoader.o -L/usr/local/cuda-11.7/lib64/
    -lcudart -lGL -lglut -lcudart
./checker.pl
Smartmatch is experimental at ./checker.pl line 75.

--------------
Running tests on ghc28.ghc.andrew.cmu.edu, size = 1150, mode = cuda
--------------

Scene : rgb
Correctness failed ... Check ./logs/correctness_rgb.log
Your time : 97.6092
Target Time: 0.1912

Scene : rgby
Correctness failed ... Check ./logs/correctness_rgby.log

Scene : rand10k
Correctness failed ... Check ./logs/correctness_rand10k.log
Your time : 22.5087
Target Time: 1.9674

Scene : rand100k
Correctness failed ... Check ./logs/correctness_rand100k.log
Your time : 502.5219
Target Time: 18.5692

Scene : biglittle
Correctness failed ... Check ./logs/correctness_biglittle.log
Your time : 283.4214
Target Time: 14.2726

Scene : littlebig
Correctness failed ... Check ./logs/correctness_littlebig.log

Scene : pattern
Correctness failed ... Check ./logs/correctness_pattern.log
Your time : 2.8770
Target Time: 0.2756

Scene : bouncingballs
Correctness passed!

Scene : hypnosis
Correctness passed!

Scene : fireworks
Correctness failed ... Check ./logs/correctness_fireworks.log

Scene : snow
Correctness passed!

Scene : snowsingle
Correctness passed!
```

```
Your time : 0.0669
Target Time: 0.1779

------------
Score table:
------------
--------------------------------------------------------------------------
| Scene Name    | Target Time   | Your Time     | Score        |
--------------------------------------------------------------------------
| rgb           | 0.1912        | 97.6092 (F)   | 0            |
| rand10k       | 1.9674        | 22.5087 (F)   | 0            |
| rand100k      | 18.5692       | 502.5219 (F)  | 0            |
| pattern       | 0.2756        | 2.8770 (F)    | 0            |
| snowsingle    | 0.1779        | 0.0669        | 12           |
| biglittle     | 14.2726       | 283.4214 (F)  | 0            |
--------------------------------------------------------------------------
|                               | Total score:  | 12/72        |
--------------------------------------------------------------------------
```

ii. Implementing the CUDA render.

- First, the render does not meet the atomicity and ordering requirements as it assign the work of rendering each circle to a single thread. The threads are not synchronized, and the result is not deterministic. For each pixel, the thread working on it might not be in the same order as it should be (the circle order provided by the circleIndex).
  Then, it came to my mind that I should deal with each pixel independently with its required order. Distribute the work of rendering each circle to the threads, then maintain an array to store the order of circles on each pixel. The second step is to distribute the work of each pixel to the threads with the order I have saved.
  This method works but is not efficient as in the first step we cannot fully utilize the GPU parallelism and the order storage will cause out of memory if the image is too large. Result is as follows.

```
siqiguo@ghc28:~/private/15-618-Parallel-Computing/asst2/render$ make check
mkdir -p objs/
g++ -m64 -O3 -Wall -g -o render objs/main.o objs/display.o objs/benchmark.o
    objs/refRenderer.o objs/cudaRenderer.o objs/noise.o objs/ppm.o objs/sceneLoader.o
    -L/usr/local/cuda-11.7/lib64/ -lcudart -lGL -lglut -lcudart
./checker.pl
Smartmatch is experimental at ./checker.pl line 75.

--------------
Running tests on ghc28.ghc.andrew.cmu.edu, size = 1150, mode = cuda
--------------

Scene : rgb
Correctness passed!
Your time : 264.9940
Target Time: 0.2034

Scene : rgby
Correctness passed!

Scene : rand10k
Correctness failed ... Check ./logs/correctness_rand10k.log
Your time : 34.1058
Target Time: 1.9736
```

```
Scene : rand100k
Correctness failed ... Check ./logs/correctness_rand100k.log
Your time : 286.9967
Target Time: 18.6039

Scene : biglittle
Correctness failed ... Check ./logs/correctness_biglittle.log
Your time : 289.9027
Target Time: 14.3019

Scene : littlebig
Correctness failed ... Check ./logs/correctness_littlebig.log

Scene : pattern
Correctness passed!
Your time : 380.7105
Target Time: 0.2827

Scene : bouncingballs
Correctness passed!

Scene : hypnosis
Correctness passed!

Scene : fireworks
Correctness passed!

Scene : snow
Correctness passed!

Scene : snowsingle
Correctness passed!
Your time : 3.3260
Target Time: 0.1975

------------
Score table:
------------
--------------------------------------------------------------------------
| Scene Name    | Target Time   | Your Time     | Score         |
--------------------------------------------------------------------------
| rgb           | 0.2034        | 264.9940      | 2             |
| rand10k       | 1.9736        | 34.1058 (F)   | 0             |
| rand100k      | 18.6039       | 286.9967 (F)  | 0             |
| pattern       | 0.2827        | 380.7105      | 2             |
| snowsingle    | 0.1975        | 3.3260        | 2             |
| biglittle     | 14.3019       | 289.9027 (F)  | 0             |
--------------------------------------------------------------------------
|                               | Total score:  | 6/72          |
--------------------------------------------------------------------------
```

- Next, optimize this method with the similar idea. Try to rely on the GPU thread parallelism to accelerate the computation.

  I gained some hints from AI, that I should partition the work of rendering the circles into some subsets of the circles and process them in batches.

  The key idea is to enable as more threads as possible to work. At the very end, each pixel should be processed by one thread. These threads should be assigned the ordering work (store circle index into thread-own buffer) evenly.

  In another view, the image is divided into blocks (corresponding to the GPU blocks), and each block can be processed by a thread block. Each thread in this block is just responsible for a subset of circles to make circle ordering work efficient. The order (circle index) stored in the thread-scope buffer will be merged to the block-scope buffer to make sure later each thread on each pixel can access to all circles in its lifespan.

```
siqiguo@ghc28:~/private/15-618-Parallel-Computing/asst2/render$ make check
mkdir -p objs/
g++ -m64 -O3 -Wall -g -o render objs/main.o objs/display.o objs/benchmark.o
    objs/refRenderer.o objs/cudaRenderer.o objs/noise.o objs/ppm.o objs/sceneLoader.o
    -L/usr/local/cuda-11.7/lib64/ -lcudart -lGL -lglut -lcudart
./checker.pl
Smartmatch is experimental at ./checker.pl line 75.

--------------
Running tests on ghc28.ghc.andrew.cmu.edu, size = 1024, mode = cuda
--------------

Scene : rgb
Correctness passed!
Your time : 0.1423
Target Time: 0.1523

Scene : rgby
Correctness passed!

Scene : rand10k
Correctness passed!
Your time : 3.7000
Target Time: 1.6040

Scene : rand100k
Correctness passed!
Your time : 34.4337
Target Time: 15.1970

Scene : biglittle
Correctness passed!
Your time : 20.0479
Target Time: 11.4324

Scene : littlebig
Correctness passed!

Scene : pattern
```

```
Correctness passed!
Your time : 0.2291
Target Time: 0.2299

Scene : bouncingballs
Correctness passed!

Scene : hypnosis
Correctness passed!

Scene : fireworks
Correctness passed!

Scene : snow
Correctness passed!

Scene : snowsingle
Correctness passed!
Your time : 0.1283
Target Time: 0.1451

------------
Score table:
------------
---------------------------------------------------------------------------
| Scene Name    | Target Time   | Your Time     | Score         |
---------------------------------------------------------------------------
| rgb           | 0.1523        | 0.1423        | 12            |
| rand10k       | 1.6040        | 3.7000        | 7             |
| rand100k      | 15.1970       | 34.4337       | 7             |
| pattern       | 0.2299        | 0.2291        | 12            |
| snowsingle    | 0.1451        | 0.1283        | 12            |
| biglittle     | 11.4324       | 20.0479       | 8             |
---------------------------------------------------------------------------
|                             | Total score:  | 58/72         |
---------------------------------------------------------------------------
```

More details about the implementation and optimization are in the code comments.