Parallel computing
Manuel Göster, Ilker Canpolat

Assignment 04: CUDA

Task 1: More OpenMP

a)

Measure the execution time for the loop which iterates over workQueue items in rasteriseCPU() to render them. Show the runtimes of this loop for a few runs of the program in your report, as well as the settings you chose..

command: `../gpurender/gpurender --width=3413 --height=1920 --depth=4`

```
real    0m33.282s
user    0m32.900s
sys     0m0.028s
```

time of the loop (3 runs):
1: 29814 ms
2: 29889 ms
3: 29816 ms

c)

Run the program a few times and look at the images it produces. Are they correct?

No, the image has some planes (rectangle & triangle) and the amount of the spheres are significantly decreased for a depth = 4.
It also seems that these planes overlap the spheres.

Are they always the same?

No, in the picture the color and the position of the spheres are different. So each run creates another picture.

Document the cause of any race conditions you found in your report.

```
Mesh &transformedMesh = transformedMeshes.at(i);
rasteriseTriangles(transformedMesh, frameBuffer, depthBuffer, width, height);
```

Passing transformedMesh as reference causes a race condition because the reference is passed to the runVertexshader method. In this method there is a loop which iterates over the vertices size. So every thread will do it too and overwrite each time the value from the transformedMesh. Passing as Copy fix this race condition.

d)

Question is missing...

```
#pragma omp parallel for
#pragma omp parallel for schedule(static,5)
#pragma omp parallel for schedule(dynamic,5)
#pragma omp parallel for schedule(guided,5)
```

e)

What are the speedups of the different scheduling strategies relative to the single threaded version? Document the runtime of each scheduling strategy and speedups relative to the single threaded variant in your report.

(after fixing race condition)
time for static: 7676 ms
time for dynamic: 7669 ms
time for guided: 7510 ms

speedup
static: 29814ms/7676ms = 3.88 times faster
dynamic: 29814ms/7669ms = 3.88 times faster
guided: 29814ms/7510ms = 3.96 times faster

f)

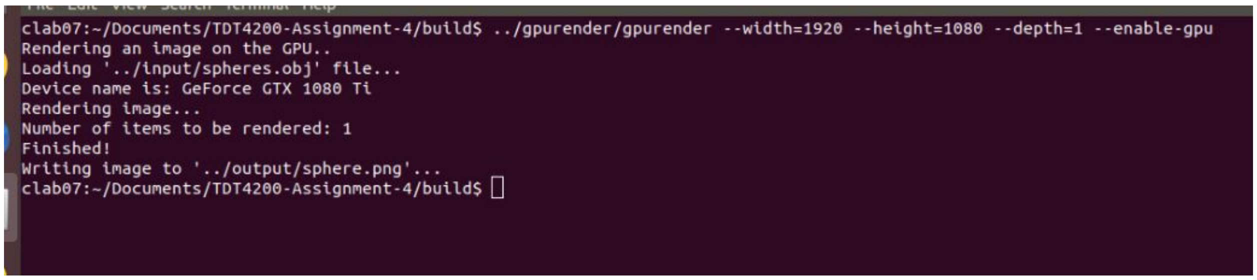
Are the runtimes you measured for the different scheduling strategies what you would expect them to be?
Yes. We expected static to be the slowest, dynamic to be second fastest and guided to be the fastest.
Can you expect one specific scheduling strategy to be faster than the others?
The scheduling type guided should be faster than static and dynamic when the chunk size is specified (e.g. 8). Because of specifying the chunk size the initial chunk should be large enough to reduce some overhead.

Task 2: Getting started

b)

Print the contents of the "name" field to the command line, and show a screenshot of it in your report.

statement = ../gpurender/gpurender --width=1920 --height=1080 --depth=1 --enable-gpu



```
clab07:~/Documents/TDT4200-Assignment-4/build$ ../gpurender/gpurender --width=1920 --height=1080 --depth=1 --enable-gpu
Rendering an image on the GPU..
Loading '../input/spheres.obj' file...
Device name is: GeForce GTX 1080 Ti
Rendering image...
Number of items to be rendered: 1
Finished!
Writing image to '../output/sphere.png'...
clab07:~/Documents/TDT4200-Assignment-4/build$
```

Task 3: Some Planning

a) Creating a Tree

```
renderMeshes()
    for 0 -> totalItemsToRender
        for 0 -> meshCount
            for 0 -> meshes[meshIndex].vertexCount / 3
                runVertexShader v0
                runVertexShader v1
                runVertexShader v2

                rasterizeTriangle

runVertexShader()
    no loop :(

rasterizeTriangle()
    for minx -> maxx
        for miny -> maxy
            runFragmentShader()

runFragmentShader()
    for 0 -> amountLightSources
```

Final tree:

```
renderMeshes()
for 0 -> totalItemsToRender
    for 0 -> meshCount
        for 0 -> meshes[meshIndex].vertexCount / 3
            for minx -> maxx
                for miny -> maxy
                    for 0 -> amountLightSources
```

loop	runtime of 1 iteration
0	4000 microseconds --> 4ms --> long
1	777 microseconds --> 0.78ms --> medium
2	1 microsecond --> short
3	0 microseconds --> short
4	0 microseconds --> short
5	0 microseconds --> short
6	0 microseconds --> short

b)

Develop a strategy for how you will divide the program between threads. Which loops will you "break apart" in order to divide them between more threads, and which will you leave intact? Briefly discuss your strategy, and support why you chose it.

The loop 0 and 1 are expensive compared to the other loops. So our strategy is to break these loops apart and leave the loops 2 - 6 intact.

Task 6: Solving Issues and Evaluating the Result

a)

There are two race conditions: We resolved the one in rasteriseTriangle

```
/* !! RACE CONDITION !! */
//depthBuffer[y * width + x] = pixelDepthConverted;
atomicExch(&depthBuffer[y * width + x], pixelDepthConverted);
```

The other one is in runFragmentShader

```
/* !! RACE CONDITION !! */
   frameBuffer[4 * baseIndex + 0] = colour.x * 255.0f;
   frameBuffer[4 * baseIndex + 1] = colour.y * 255.0f;
   frameBuffer[4 * baseIndex + 2] = colour.z * 255.0f;
   frameBuffer[4 * baseIndex + 3] = 255;
```

But can`t be solved that easily as the first one, as there is no atomicExch for unsigned char defined. (see: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions)
It would be possible to solve it e.g. by using a mutex.

b)

Measure the runtime of your kernel. When rendering an image using the same settings as the ones you used for the OpenMP measurements, what is the speedup you achieved over the single-threaded CPU and multi-threaded CPU variants?
command: `gpurender/gpurender --width=3413 --height=1920 --depth=4`

GPU runtime of kernel: 4 microseconds → 0.004ms
Single core: 1: 29814 ms
OpenMP: time for guided: 7510 ms
Speedup, compared to single thread: 29814ms / 0.004ms = 7453500 times faster
Speedup, compared to OpenMP implementation: 7510ms / 0.004ms = 1877500 times faster

What is the achieved speedup when you also include the time spent allocating and copying buffers to and from the GPU?
GPU runtime: 385400 microseconds → 385ms
Speedup, compared to single thread: 29814ms / 385ms = 77.04 times faster
Speedup, compared to OpenMP implementation: 7510ms / 385ms = 19.05 times faster