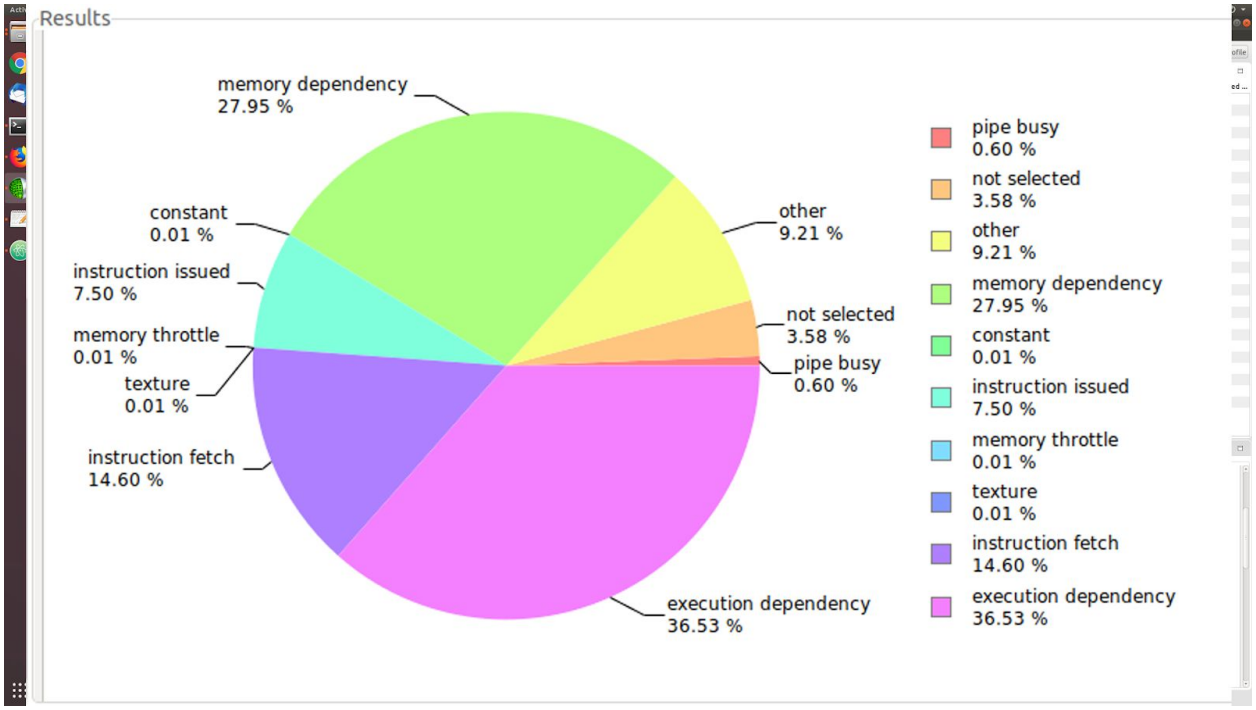
Parallel computing
Manuel Göster, Ilker Canpolat

Assignment 05: More on CUDA

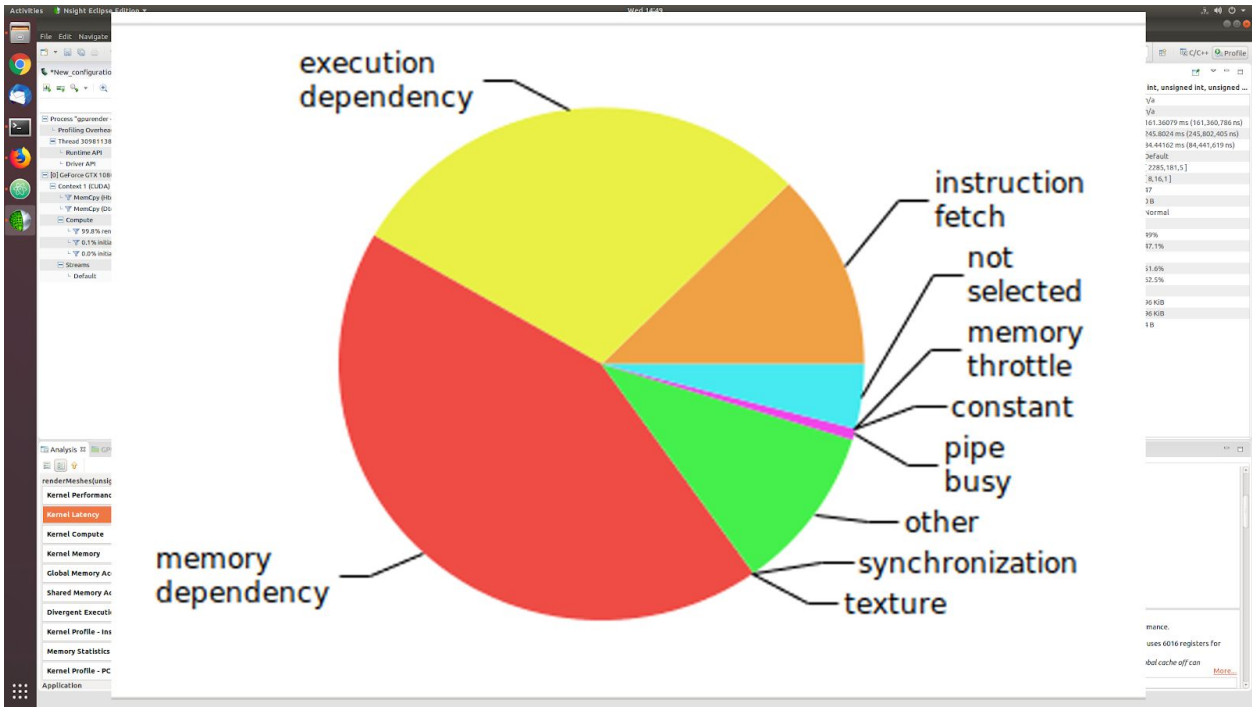
Task 1: For each of the tasks below, use the profiler to determine the requested value, and indicate where you found it (either a screenshot or a single sentence)

a) What is on average the percentage of the time in which warps are executing instructions (not being blocked)?

7.50 % : instruction issued → those are not blocked

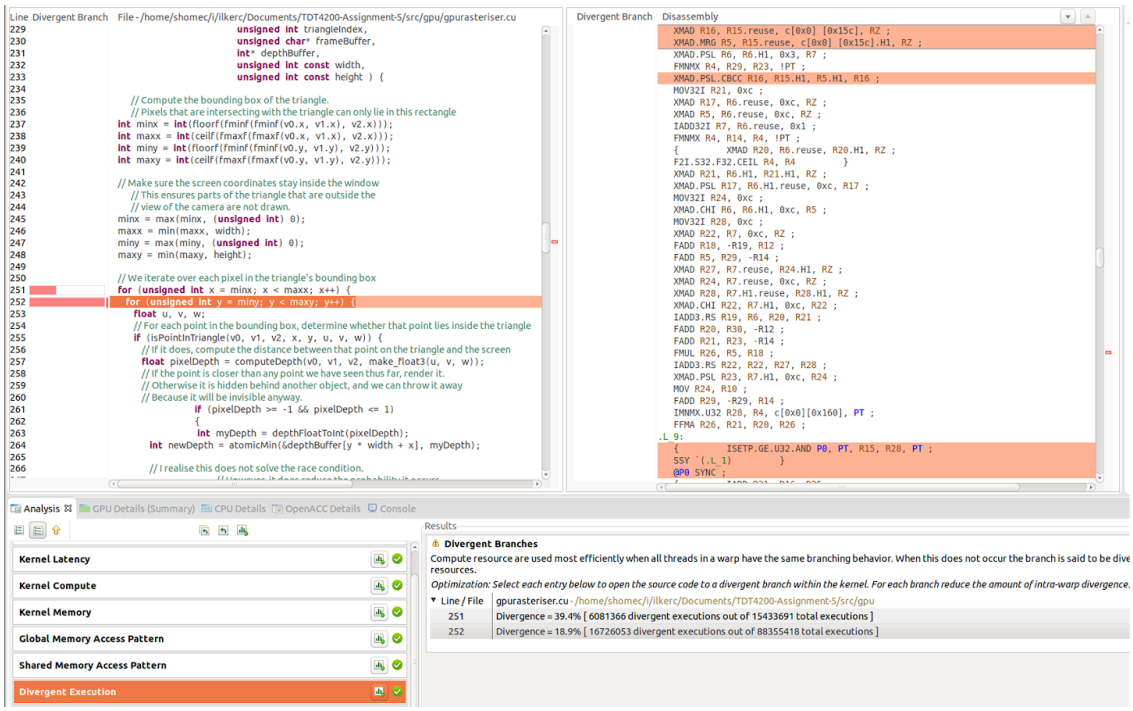


b) What percentage of cycles where threads are blocked is due to a memory request being processed?



Memory dependency: about 45%

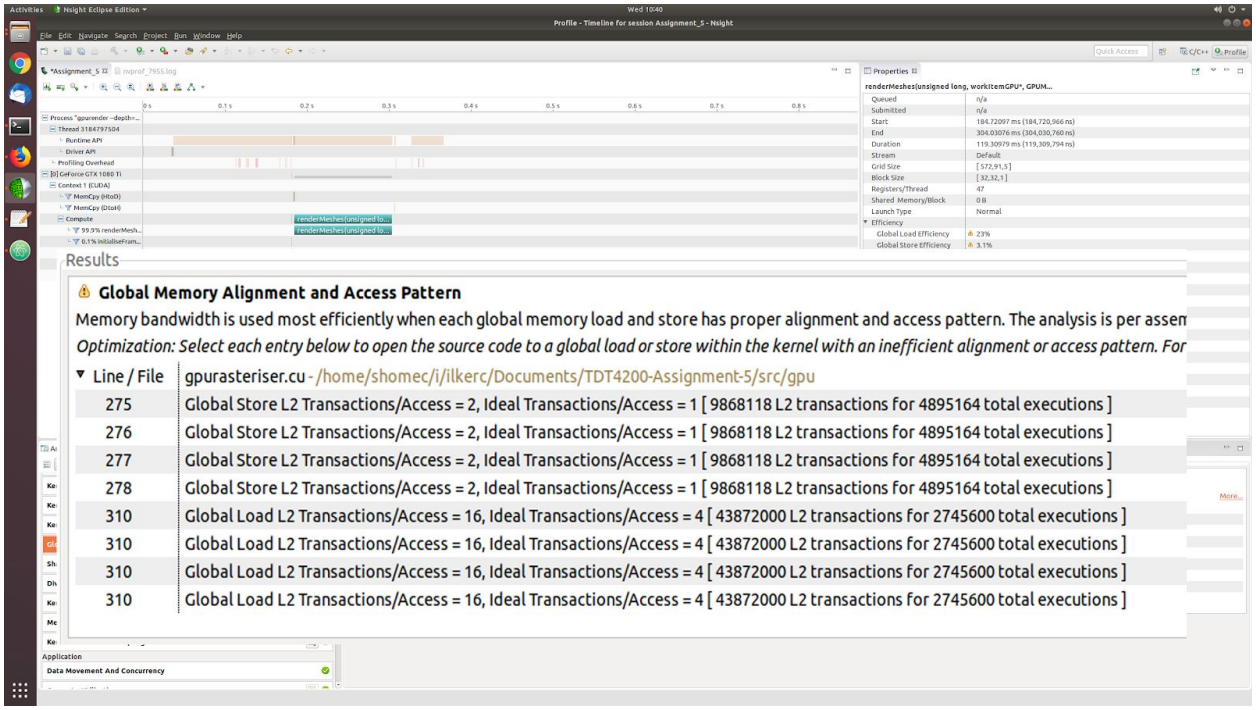
c) Which section of code is generally executed by the fewest number of threads in a warp?



Its the code section starting at line 256 after the if statement.

d) Which line(s) of code have the worst memory access pattern?

Line 310



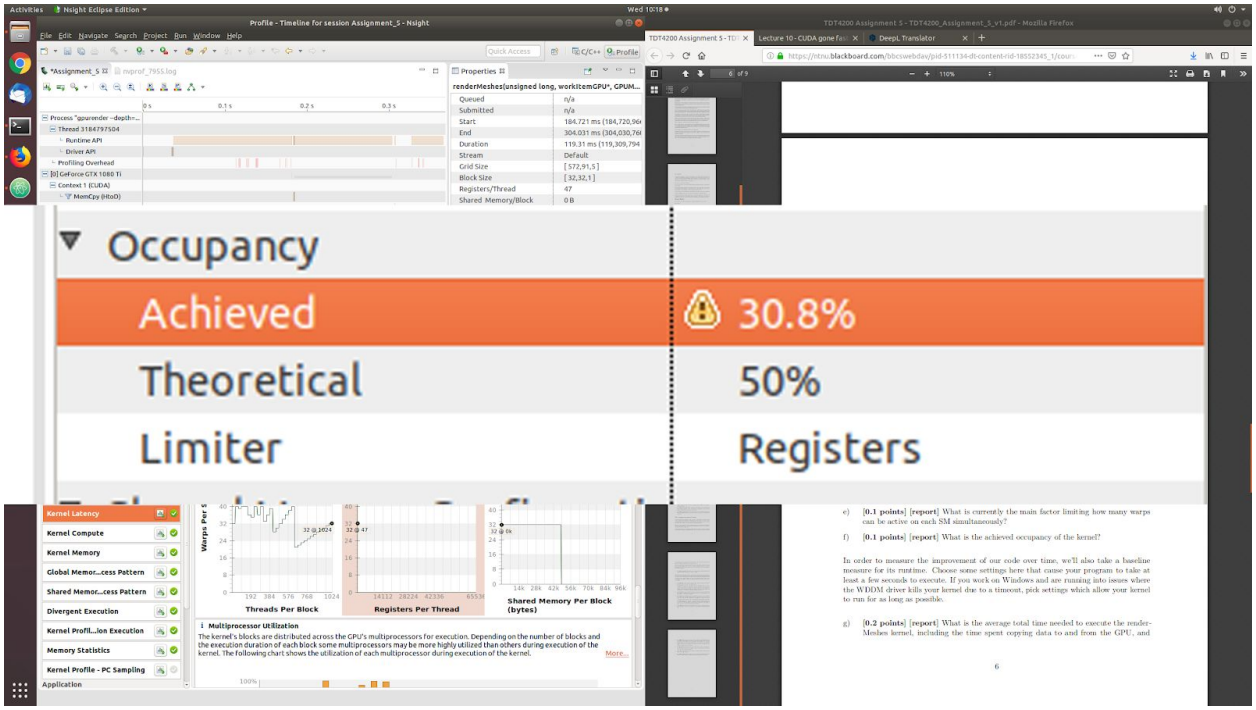
e) What is currently the main factor limiting how many warps can be active on each SM simultaneously?

The block size 32,32,1

f) What is the achieved occupancy of the kernel?

Parallel computing
Manuel Göster, Ilker Canpolat

30.8%



g) What is the average total time needed to execute the render-
Meshes kernel, including the time spent copying data to and from the GPU, and running the buffer initialisation kernels?

Setting: 1920 X 1080, depth = 4

average time: 120.728297 ms
execution of renderMeshes Kernel: 119.30979 ms
Copy to HostToDevice: 0.073602 ms
copy to deviceToHost: 1.24311 ms
Framebuffer initialization: 0.081666 ms
Depthbuffer initialization: 0.020129 ms

Task 2: Low-hanging Fruit

a) Define the term “Occupancy”. How does better occupancy generally lead to better performance?

The term "Occupancy" describes the ratio of executing warps in respect to the total number of cycles. A better Occupancy indicates that the hardware uses more of their capability to hide latency. So it also means there are less cycles that leave the hardware unused.

b) What are the main ways in which the number of warps which can be active simultaneously on an SM can be limited?

It depends on

- the number of registers needed per thread, because the size of the SM register file is limited and shared by all threads within this SM
- the block size, because warps are group in blocks. Thus, the amount of warps in a SM is not optimal, because there is not enough memory in the SM register file left to load a full block of warps into the SM
- Shared memory usage

(see: p.44-46 Lecture 10)

c) What does “Latency Hiding” mean, and how does it relate to occupancy?

Latency hiding means that warps can be reordered to allow ready warps to execute. This means if a warp has to wait for memory transactions, other warps can continue their execution while the warp is waiting for memory. Thus, the waiting times are productively used by executing other warps. With a good occupancy all clock cycles use its full capability. To achieve this free slots can be filled with switching warps.

d) Improve the occupancy of the handout code by improving the memory access pattern of the line(s) you found in task 1d). What is the occupancy of the kernel now?

Changed threadsPerWorkQueueBlock from 32 to 8 and threadsPerVertexBlock from 32 to 16. Thus the memory access pattern is improved.

Now, the Occupancy of the kernel is: 50.5%

e) Improve the occupancy of the handout code by modifying the kernel launch parameters. Try at least 5 different configurations, listing the used parameters and achieved occupancy in your report.

Occupancy: 50.5%: threadsPerWorkQueueBlock = 8; threadsPerVertexBlock = 16;

Occupancy: 45.6% : threadsPerWorkQueueBlock = 8; threadsPerVertexBlock = 8;

Occupancy: 46.7%: threadsPerWorkQueueBlock = 8; threadsPerVertexBlock = 24;

Occupancy: 44.3%: threadsPerWorkQueueBlock = 16; threadsPerVertexBlock = 16;

Occupancy: 45.8%: threadsPerWorkQueueBlock = 4; threadsPerVertexBlock = 16;

f) Measure the execution time of the renderMeshes kernel again. What is the speedup of the changes you’ve made thus far, relative to your measurement in task 1g)?

Setting: 1920 X 1080, depth = 4

average time: 70.507168 ms

execution of renderMeshes Kernel: 69.60733 ms

Copy to HostToDevice: 0.067107 ms

copy to deviceToHost: 0.729431 ms

Framebuffer initialization: 0.081667 ms

Depthbuffer initialization: 0.021633 ms

Speedup: 120.728297/70.507168 = 1.7123 times faster

Task 3:

NOTE:
The subtask e and f can not be tested because of some “kernel crashes” according to the student assistant. Even print statements do not work. So this is reason why we can not also measure the time for the subtask i and j. Sorry for this!

Our idea for the task 3 was to create a for loop which iterates over the amount of pixels of bounding box. In this loop every thread should calculate 1 pixel of the image. To have the slope for the x-coordinate we are using the modulo operator to stay in the bounding width. The slope of the y-coordinate is created by using the division operator. Our loop variable will be added with the threadIdx.x.

```
for (unsigned int i = 0; i < amount_of_pixels_of_bounding_box; i += 32) {
        unsigned int x = minxlol + (i + threadIdx.x) % boundingWidth;
        unsigned int y = minylol + (i + threadIdx.x) / boundingWidth;
        rasteriseSinglePixel(v0lol, v1lol, v2lol, x, y,
                mesh, triangleIdx, frameBuffer, depthBuffer, width);
}
```

a) Why should we care about whether individual threads take significantly longer than average to render a triangle?

It is important that threads within a warp have about the same execution time as they have to wait for each other. If one thread takes significantly longer than the others within a warp, all the other 31 threads will have to wait for the one thread.

i) see Note

j) see Note