COMP.CE.350 Multicore and GPU Programming Project Work - part 2

Eetu Soronen - 152830694

Performance measurement table

tested and benchmarked on my personal system (RTX 4070 and AMD Ryzen 7800X3D)

Benchmarks with 1920 x 1080 window size	Clang Release
The Original C Version on CPU without optimizations	742ms
The Original C version on CPU with best optimizations (“-ffast-math”)	572ms
The OpenCL version with GraphicsEngine on GPU, WG size 1x1	35ms
The OpenCL version with GraphicsEngine on GPU, WG size 4x4	15ms
The OpenCL version with GraphicsEngine on GPU, WG size 8x4	14ms
The OpenCL version with GraphicsEngine on GPU, WG size 8x8	16ms
The OpenCL version with GraphicsEngine on GPU, WG size 16x16	15ms
The OpenCL version with GraphicsEngine on GPU, WG size 16x16 (with no –ffast-math)	16ms
Using OpenCL for both physics and graphicsEngines (satellite count 64)	48ms
Using OpenCL for both physics and graphicsEngines (satellite count 1024)	65ms

I made a kernel function for physicsEngine as well, but even when removing the unecessary back-and-forth writes between CPU and GPU, the latency was still ~50ms. The physics kernel can be run by setting “#define USE_PHYSICS_KERNEL 1” in constants.h. On the upside changing satellite count from 64 -> 1024 increases latency only by ~20 ms.

Running my program on TC303 lab computers was little faster than on my own, but not significantly

Work groups above 32x32 crash my program since that is the maximum my GPU can physically support. Changing it to exotic numbers like 13x7, that aren’t a ratio of window size makes some threads to calculate pixels out of bounds and my kernel exits in those cases. The program doesn’t crash though

Otherwise only the 1x1 wg size was slower than the rest which were all practically identically fast. This is likely because more work groups mean more overhead in creating and managing them. As for why there is no speed bump after 1x1 WG size, I think I’m CPU bottlenecked which means that most of the time is spent

calculating the physics. In theory though less work groups would mean better utilization of GPU's parallel architecture and less synchronization overhead.

Benchmarks with 80 x 80 window size	Clang Release
The Original C version on CPU with best optimizations ("-ffast-math")	91ms
The OpenCL version with graphicsengine on GPU, WG size 16x16	15ms

When window size gets reduced, the original unparallelized CPU version performance increases drastically while the GPU version changes not at all. This is further proof that my program is CPU constrained. The parallelized physics routine takes some few milliseconds and reducing the number of pixels to doesn't change the speed measurably.

Bonus question: I already did my optimizations like removing sqrts and simplifying calculations in the OpenMP portion, and i'm sure those still provide some benefit on the gpu, though probably not as much.

8. Feedback

1. I liked the first part with openmp a lot, as well as simplifying the physics and graphics engine calculations.
2. For me replacing the custom command to copy parallel.cl with this made it so it is copied even if source files are not modified:

```
add_custom_command(
    OUTPUT "${CMAKE_BINARY_DIR}/parallel.cl"
    COMMAND ${CMAKE_COMMAND} -E copy
    "${CMAKE_SOURCE_DIR}/parallel.cl" "${CMAKE_BINARY_DIR}"
    DEPENDS "${CMAKE_SOURCE_DIR}/parallel.cl"
    VERBATIM
)

add_custom_target(copy_parallel_cl ALL DEPENDS "${CMAKE_BINARY_DIR}/parallel.cl")
add_dependencies(parallel copy_parallel_cl)
```

3. OpenMP was great and interesting to learn about. GPUs even more so but also very difficult
4. For me buffers were a new concept, and understanding and getting them to not go out of bounds took the longest time. I also tried to write the initial boilerplate myself but in the end shamelessly copied it since I didn't get it working. This was also my first C / C++ project.
5. ~30h