Week 2

Hao Mao

This week mainly focused on reviewing the code in the homework assignments and clarifying some topics that were previously overlooked.

Firstly, regarding the testing section, I briefly reviewed the implementation of unit tests in the homework code.

I learned how to use the Catch2 testing framework, which is now updated to Catch3.

From the documentation, I grasped the concept of Behavior-Driven Development (BDD). Catch provides this syntax, allowing developers to describe test cases using the keywords SCENARIO, GIVEN, WHEN, THEN, and AND.

For example:

```
SCENARIO("Vector resizing affects size and capacity", "[vector]") {
    GIVEN("A vector with some items") {
        std::vector<int> v(5);

        REQUIRE(v.size() == 5);
        REQUIRE(v.capacity() >= 5);

        WHEN("the size is increased") {
            v.resize(10);

            THEN("the size and capacity change") {
                REQUIRE(v.size() == 10);
                REQUIRE(v.capacity() >= 10);
            }
        }
        WHEN("the size is reduced") {
            v.resize(0);

            THEN("the size changes but not capacity") {
                REQUIRE(v.size() == 0);
                REQUIRE(v.capacity() >= 5);
            }
        }
```

```
    }
}
```

However, it's worth noting that Catch2's assertion macros are not thread-safe, but this doesn't mean they can't be used in test cases involving threads. Just ensure that only one thread interacts with the assertion macros at any given time.

The unit tests in the homework code were traditionally completed using template functions and Functors.

I learned a common practice to prevent variables from being optimized away by the compiler, which is to use the variable at least once, like returning it. Also, for small data blocks, performance testing can be more volatile, so it is necessary to increase the number of tests to ensure data reliability.

I ran a roofline test on my machine, and by analyzing the resulting data, I could estimate the size of L1 cache. For example, when the working set size exceeds the capacity of a certain level of cache, you'll notice a significant drop in performance.

However, I'm not sure if running this program in Docker would yield an accurate estimation of the L1 cache size.

I discovered that Apple's M1 chip has a slightly different cache design compared to x86, like my Intel Core i5-13600K. The M1's L1 cache consists of 128 KB of instruction cache and 64 KB of data cache per core, L2 cache is shared across clusters of four cores with 12 MB, and L3 cache is 16 MB shared across the whole chip.

Meanwhile, the i5-13600K features 32 KB of L1 instruction cache and 48 KB of L1 data cache per P-core, 1.25 MB of L2, and a shared 24 MB of L3 cache.

I also learned about three methods of storing RGB data (AOS, SOA, Tensor) and analyzed the pros and cons of each. For instance, SOA might be a better choice for highly parallel processing required in graphics rendering or video processing; tensors are more suitable for applications involving complex data models and multi-dimensional data analysis in machine learning. AOS, being straightforward and intuitive, fits most traditional applications where extreme performance demands are not an issue.

Additionally, I examined the impact of the order of ijk loops on the performance during matrix multiplication.