mp6 report

name: Yao Xinchen

zju id: 3220116167

intl id: xinchen.22@intl.zju.edu.cn

netid: yao29

analysis of the original

To find out which function take majority time, I measured how long each part take by setting the following labels.

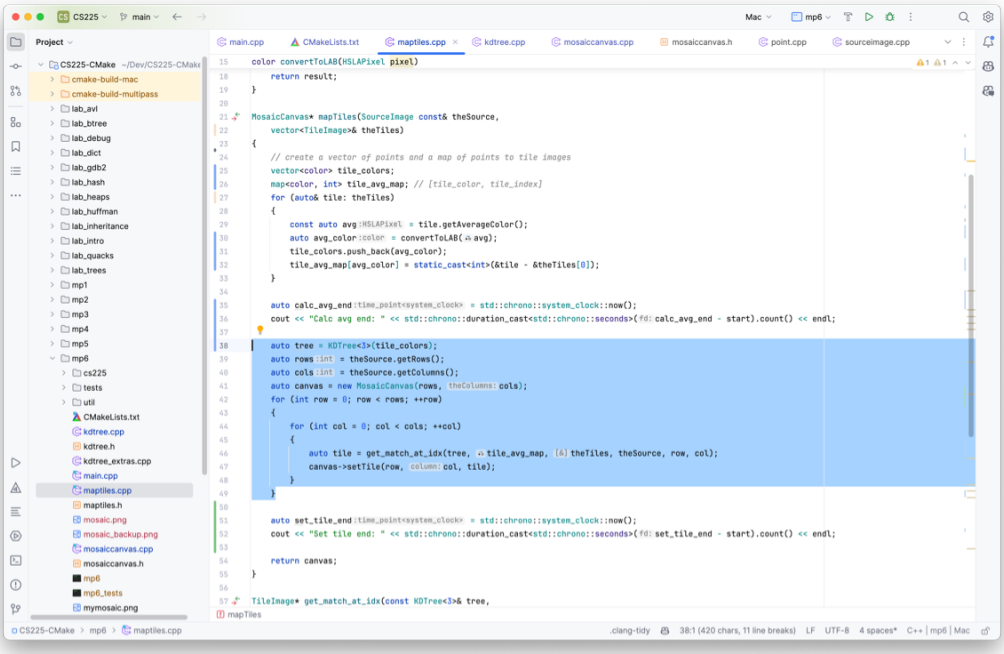
```
chrono::time_point<chrono::system_clock> start; // global

auto time_point = std::chrono::system_clock::now();
cout << "Time point: "
     << std::chrono::duration_cast<std::chrono::seconds>(draw_end - start).count()
     << endl;
```

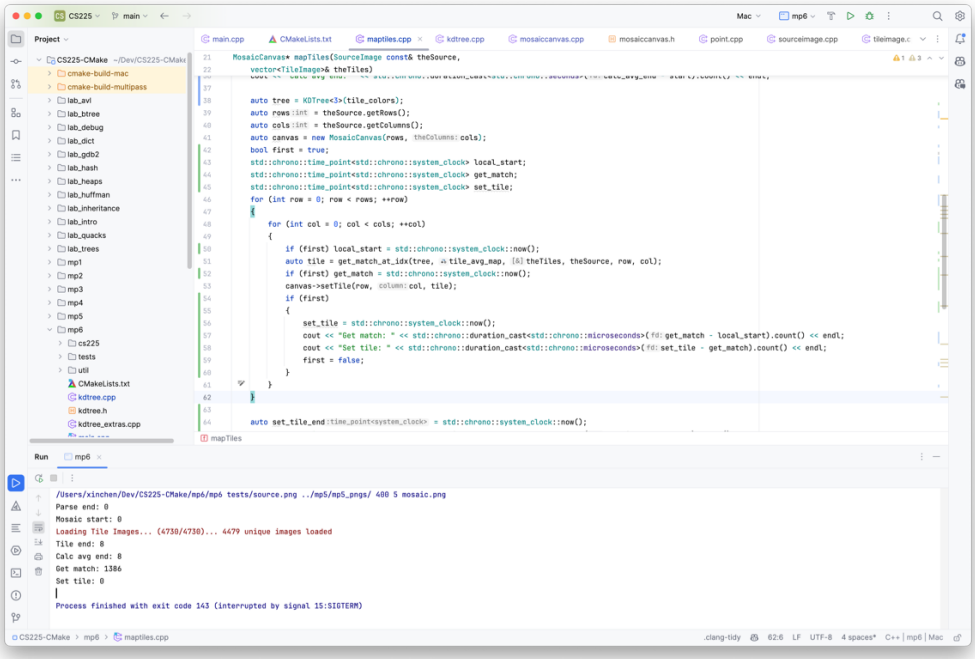
The outcome is

```
Parse end: 0
Mosaic start: 0
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Tile end: 8
Calc avg end: 8
Set tile end: 297
Map end: 297
Saving Output Image... Done
Draw end: 299
Mosaic end: 299
Elapsed time: 299s
```

Therefor, a part in the function `MosaicCanvas* mapTiles(SourceImage const& theSource, vector& theTiles)` takes the most time.



With further measurement



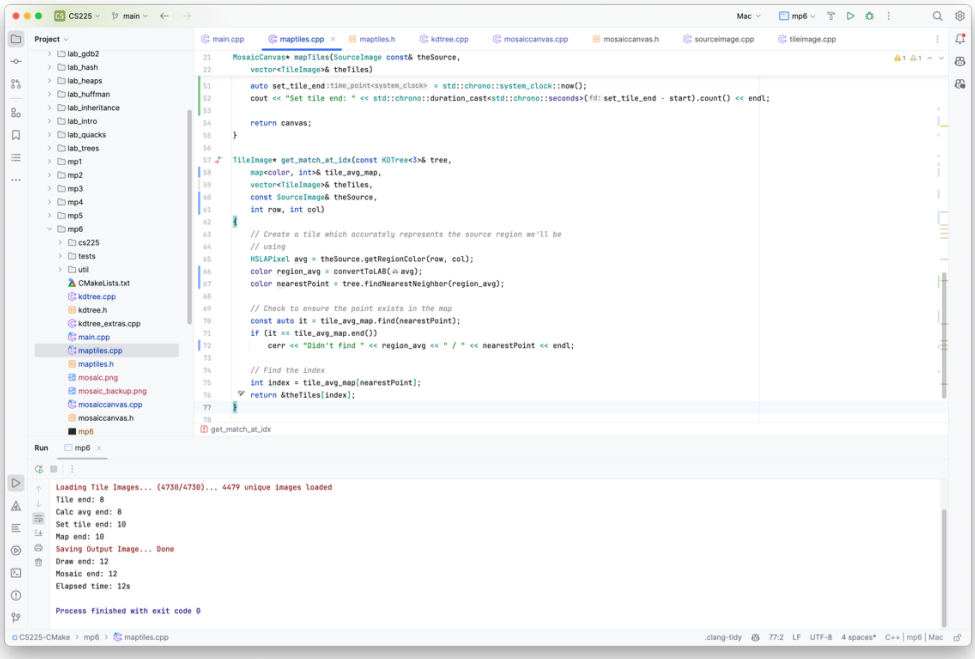
Each `get_match_at_idx()` takes 1386 microseconds.

Then I noticed that the `tile_avg_map` is not passed with reference. So, it would be copied, each time `get_match_at_idx()` is invoked. This would occupy a large amount of memory and take much time.

analysis of the new

reference

After I changed it to reference, the whole program takes 12 seconds.



parrellel computing

At the same time, there are many loops used.

1. loading the tiles from files
2. calculating average color of each tile
3. mapping tiles to each region
4. copying the tiles to the output image

The order of each loop body doesn't matter. So, the loops can be optimized with parrellel computing.

Here is an example of how to load the tiles.

```
void loadImages(const vector<string>& imageFiles, size_t start, size_t end,
    vector<TileImage>& images, set<HSLAPixel>& avgColors)
{
    for (size_t i = start; i < end; ++i)
    {
        cerr << "\rLoading Tile Images... ("
                << (i + 1) << "/" << imageFiles.size()
                << ")" << string(20, ' ') << "\r";
        cerr.flush();
        PNG png;
        png.readFromFile(imageFiles.at(i));
        TileImage next(png);
        mtx.lock(); // entering critical section
```

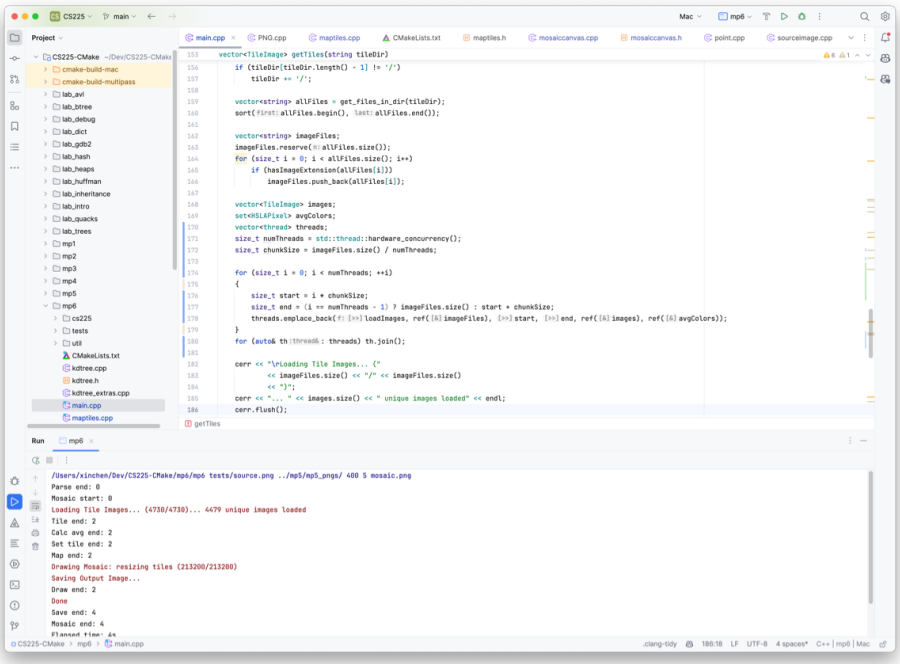
```
        if (avgColors.count(next.getAverageColor()) == 0)
        {
            avgColors.insert(next.getAverageColor());
            images.push_back(next);
        }
        mtx.unlock(); // leaving critical section
    }
}

vector<TileImage> getTiles(string tileDir)
{
    // ...
    vector<TileImage> images;
    set<HSLAPixel> avgColors;
    vector<thread> threads;
    size_t numThreads = std::thread::hardware_concurrency();
    size_t chunkSize = imageFiles.size() / numThreads;

    for (size_t i = 0; i < numThreads; ++i)
    {
        size_t start = i * chunkSize;
        size_t end = (i == numThreads - 1) ? imageFiles.size() : start + chunkSize;
        threads.emplace_back(loadImages, ref(imageFiles), start, end, ref(images),
ref(avgColors));
    }
    for (auto& th: threads) th.join();

    // ...
}
```

After applying this to the loops, the total time used is minimized to 4 seconds.



The remaining time is mostly used on reading and writing to memory.