

## mp6 report

---

name: Yao Xincheng

zju id: 3220116167

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

netid: yao29

## analysis of the original

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

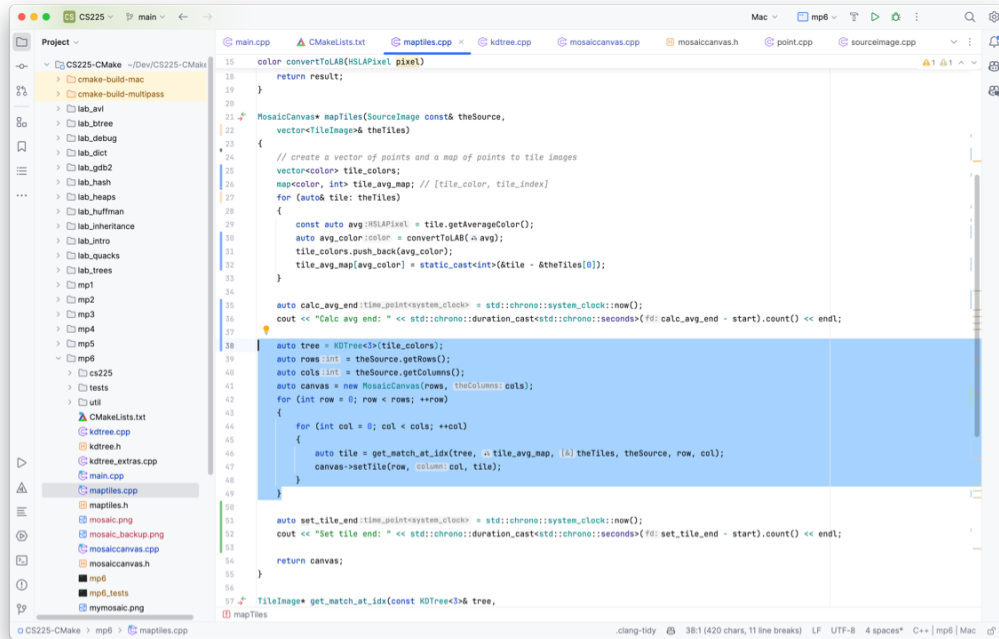
```
chrono::time_point<chrono::system_clock> start; // global

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

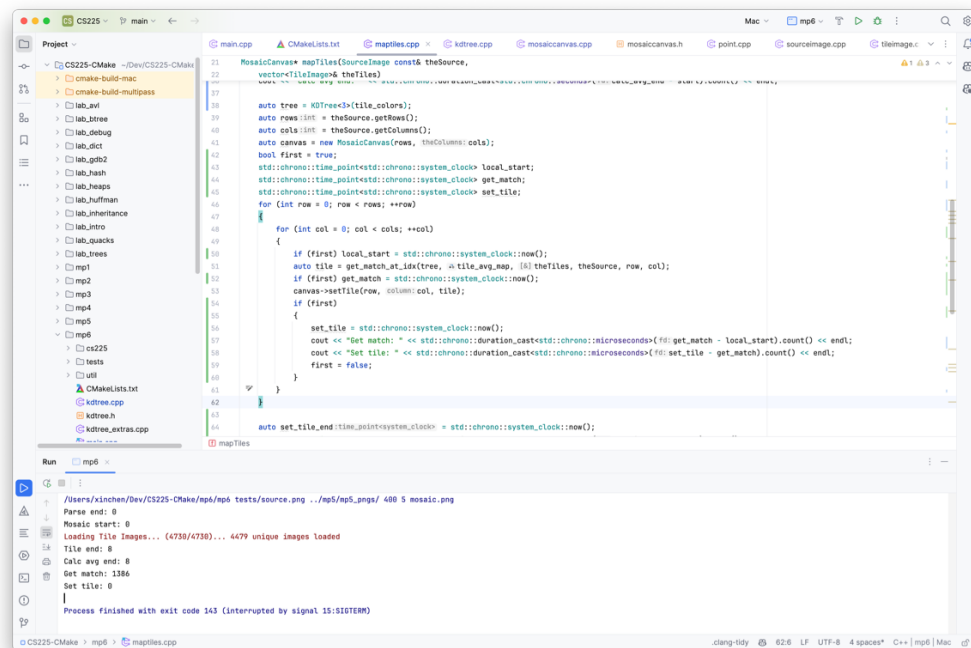
The outcome is

```
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
```

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



With further measurement



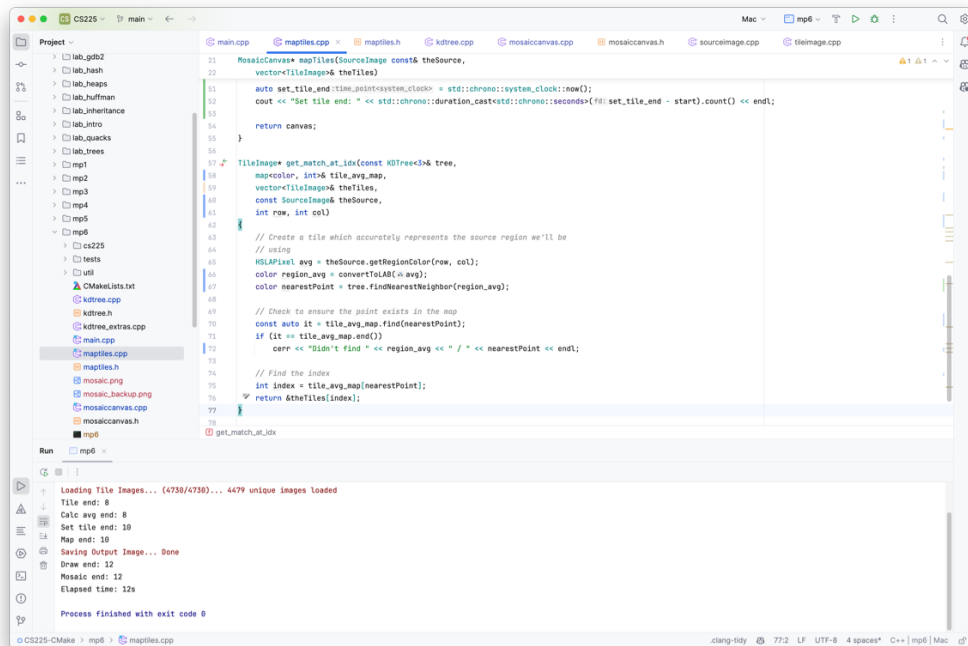
Each `get_match_at_idx()` takes 1386 microseconds.

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.



## parallel 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 parallel 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> getFiles(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.

```

153 vector<TileImage> getFiles(string tileDir)
154 {
155     if (tileDir[tileDir.length() - 1] != '/')
156         tileDir += '/';
157
158     vector<string> allFiles = get_files_in_dir(tileDir);
159     sort(first: allFiles.begin(), last: allFiles.end());
160
161     vector<string> imageFiles;
162     imageFiles.reserve(allFiles.size());
163     for (size_t i = 0; i < allFiles.size(); ++i)
164         if (hasImageExtension(allFiles[i]))
165             imageFiles.push_back(allFiles[i]);
166
167     vector<TileImage> images;
168     set<HSLAPixel> avgColors;
169     vector<thread> threads;
170     size_t numThreads = std::thread::hardware_concurrency();
171     size_t chunkSize = imageFiles.size() / numThreads;
172
173     for (size_t i = 0; i < numThreads; ++i)
174     {
175         size_t start = i * chunkSize;
176         size_t end = (i == numThreads - 1) ? imageFiles.size() : start + chunkSize;
177         threads.emplace_back(loadImages, ref(imageFiles), start, end, ref(images), ref(avgColors));
178     }
179     for (auto& th: threads) th.join();
180
181     cerr << "Loading Tile Images... (" << imageFiles.size() << " unique images loaded" << endl;
182     cerr << "Done" << endl;
183     return images;
184 }

```

```

Run mp6
~/Users/finchen/Dev/CS225-CHake/mp6/mp6 tests/source.png ../mp5/mp5.png/ 400 5 mosaic.png
Parse end: 0
Mosaic start: 0
Loading Tile Images... (4730/4730)... 4479 unique images loaded
Tile end: 2
Calc avg end: 2
Set tile end: 2
Map end: 2
Drawing Mosaic: resizing tiles (213200/213200)
Saving Output Image...
Draw end: 2
Done
Save end: 4
Mosaic end: 4
Elapsed Time: 4s

```

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