Report of MP6

ZJU ID: 3210112118 intl ID: boyao.21 NetID: boyaow2

A. Analysis of Memory Allocation and Running Time in Original MP5 Implementation

a. analysis of memory allocation

For observation 1, I find that one possible crash could be caused when generating a large mosaic because if the function setTile in the file maptiles.cpp takes the copy of a tile image as a parameter, w*h*PNG additional memory are allocated as the mosaic is populated with tile images, where w and h represent the number of “tile images” that make up the width and height of the mosaic. And this additional memory could be 1.25 TB when w=h=500 and each PNG takes about 5MB memory.

For observation 2, I find that when using in the class TileImage in the file tileimage.h, it includes two class member PNG image_ and PNG resizd_. And in this file the tile image is always resized in the function paste while the original PNG image_ is not used and not deleted, which cause some additional memory allocation.

b. analysis of running time

For observation 3, I find that for drawing the mosaic images, it runs in O(w*h*w’*h’) time, where w’ and h’ represent the width and height of the tile images as it needs to deal with each tile image’s each pixel in the corresponding place in the source image. So the running time is effectively a quartic polynomial and it could be too slow to handle a large image with large tile images.

For observation 4, I find that it in the function get_match_at_idx in the file maptiles.cpp can has a better running time. Because there are lots of data pairs in the map tile_avg_map, and it will take some additional time to make a copy of the whole map to the stack when calling the function get_match_at_idx.

B. Analysis of Memory Allocation and Running Time in New MP6 Implementation

a. analysis of memory allocation

For observation 1, by using the pointer of tile images to the helper function setTile in the function mapTiles in the file maptiles.cpp, the additional memory allocation will become w*h*C, where C is the memory allocation of a pointer, that is, 8 bytes, so that the additional memory is much smaller, that is, only 2MB when w=h=500.

For observation 2, by adding one custom constructor of the class TileImage and modifying some relevant functions' signature in the file tileimage.cpp, tileimage.h, main.cpp, I achieve that only one of the PNG member in the class TileImage will work(actually, that is always the resized_ that works). And the other PNG member is deleted. In this way, the memory allocation is decreased.

b. analysis of running time

For observation 3, by firstly finding the average color of each tile image and pushing them into a vector to build a kd-tree, the running time is O(n*w'*h') and secondly for each tile image place in the source image, we use the kd-tree to find a nearest average color tile image, the running time is O(w*h). So the total running time is O(w*h + n*w'*h'), which is much faster than the original one.

For observation 4, by passing the reference of the map to the function get_match_at_idx, the time to make a copy of the map consisting of many data pairs is saved. As the result, the total running time will be decreased as there is no need to copy the map which consists of lots of data pairs.

C. Description of Changes Made to Reduce Memory Footprint and Running Time

For observation 1, the change I made is to pass the pointer of tile images instead of passing the copy of tile images to the helper function setTile in the function mapTiles in the file maptiles.cpp, which sufficiently reduces the memory allocation. The original performance of the program is that it can not handle all these tile images and crashes. After the improvement, the program can run in an affordable memory allocation. Because this change has been made in the original mp5, so I just prove the correctness of the improvement memory allocation without showing the corresponding memory allocation data.

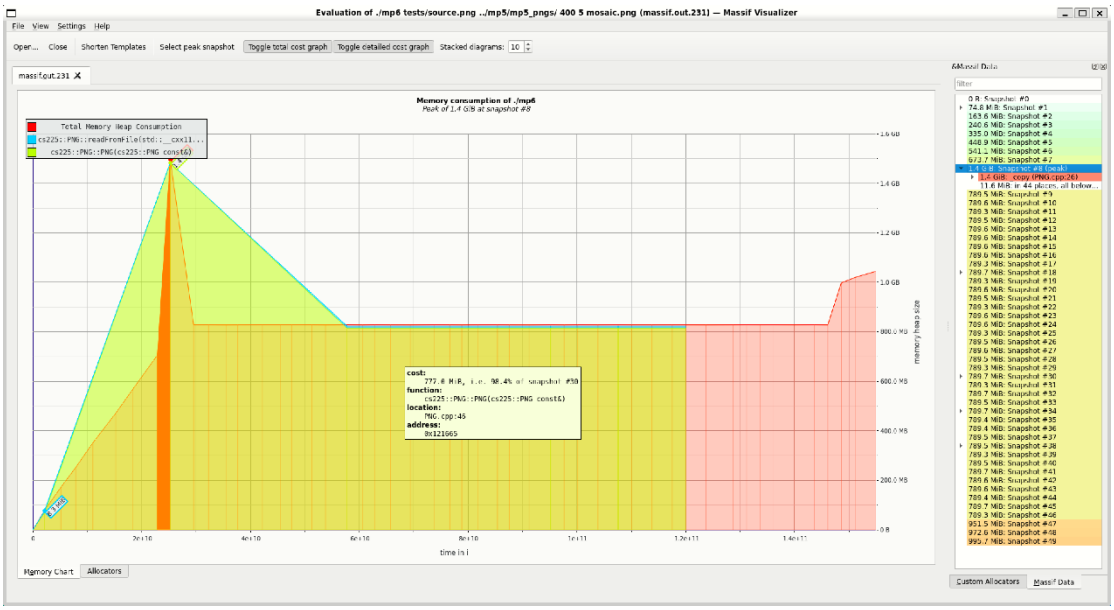

Figure 1. Original Memory Allocation when both PNG in the Class Exists

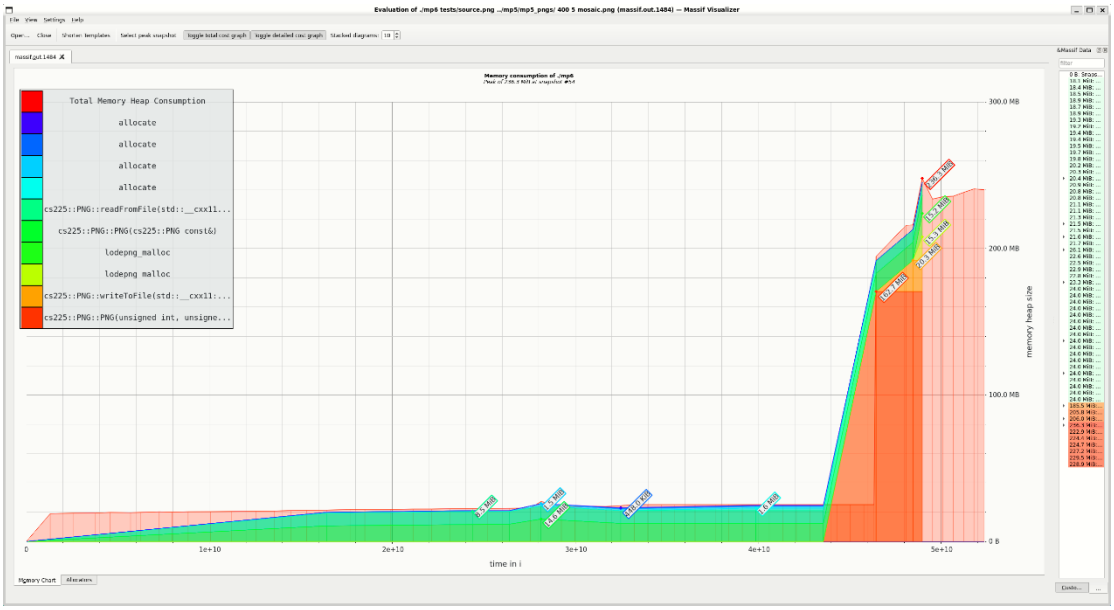

Figure 2. Improved Memory Allocation when One of PNG in the Class is Deleted

For observation 2, the change I made is to add a custom constructor in the class TileImage which will always resize the tile image and delete the other PNG image_, which decreases the memory allocation. I also modify the relevant functions' signature so that the added custom constructor will be used. And I run the code "valgrind --tool=massif ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png", "massif-visualizer massif.out.231" and "massif-visualizer massif.out.1484"(where 231 and 1484 comes from the generated file names) to find the changes to the program's memory allocation. As a result, the memory allocation of the program changes from

figure 1 to figure 2, which indicates that the memory allocation is sufficiently decreased, which confirms my previous analysis.

For observation 3, the change I made is to build a kd-tree based on the average color of each tile image and find the nearest average color tile image for each place in the source image using the kd-tree in the function mapTiles in the file maptiles.cpp, which sufficiently reduces the running time. The original performance of the program is that it takes too much time to copy each pixel to the source image. After the improvement, the program can run in an affordable running time. Because this change has been made in the original mp5, so I just prove the correctness of the improvement running time without showing the corresponding running time data.

Figure 3. Original Running Time when Passing the Copy of a Map to the Function

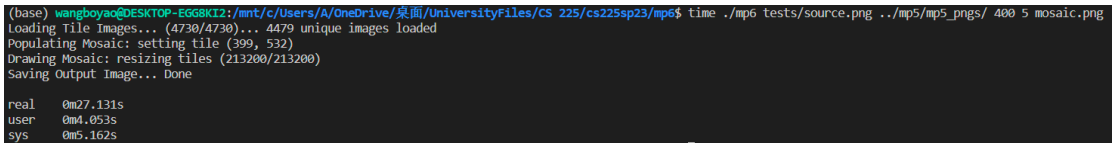

Figure 4. Improved Running Time when Passing the Reference of a Map to the Function

For observation 4, the change I made is to pass the reference of the map instead of the copy of the map to the function get_match_at_idx in the file maptiles.cpp. And I run the command “time ./mp6 tests/source.png ../mp5/mp5_pngs/ 400 5 mosaic.png” to find the changes to the program’s running time. The result is that the running time changes from figure 3 to figure 4, which indicates the running time for user changes to the one fifth of the original and a nearly half decrease for the real part, which confirms my previous analysis.