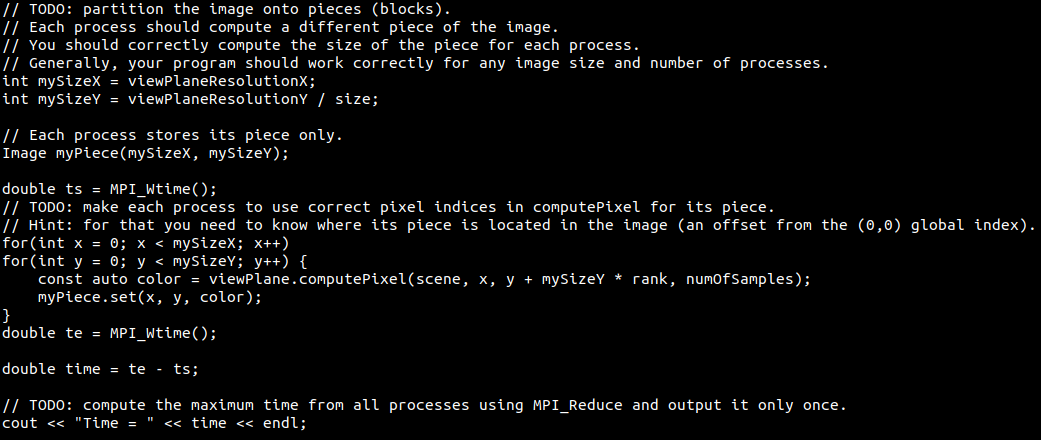
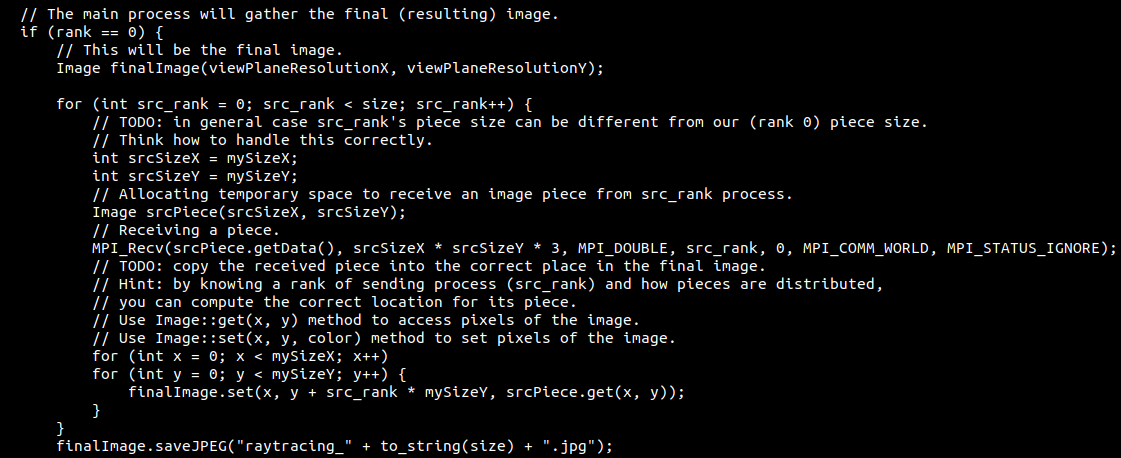
Task 4

Step 2. Implement parallel program with MPI.

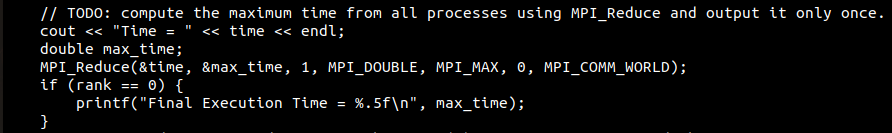
Compute block size & create peace of image & fill the image with color:



Receive piece of image and assign it’s part to the whole image:



Step 3. Study performance of your parallel program.



Default parameters to test execution time: 1280x720, 3 samples

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| NumOfThreads | Run #1 | Run #2 | Run #3 | Run #4 | Run #5 | MinTime, s | SpeedUp, % | Efficiency, % |
| 1 | 17.01 | 17.06 | 17.03 | 17.1 | 17.01 | 17.01 | 100 | 100 |
| 2 | 10.99 | 11.05 | 11.02 | 11.05 | 10.97 | 10.97 | 155 | 78 |
| 4 | 1.96 | 1.96 | 1.98 | 1.95 | 1.96 | 1.95 | 872 | 218 |
| 8 | 1.69 | 1.69 | 1.68 | 1.69 | 1.68 | 1.68 | 1013 | 127 |
| 16 | 1.11 | 1.01 | 1.07 | 1.1 | 1.55 | 1.01 | 1684 | 105 |

Step 4: Commit and push your changes to the Gitlab server

Github repo: https://github.com/MTurtle-1/HPC\_TASK-4.git

Step 5: Conclusion in a free form

The static scheduling algorithm demonstrated poor results in single thread. It can be caused by several factors: in single thread there is an unnecessary operation of creating a part of an image when the part itself is the whole image. Therefore, algorithm waste additional time to retrieve that image and copy colors from piece to the final image.

For 4, 8 and 16 threads, the execution time falls sharply and become similar to the results of static scheduling algorithms in Task 2 and Task 3. That drop in execution time cause the efficiency exceeding 100%.