

Report on task 4

Name: Amir Yakubov

Group: BDA-2008

E-mail: 201463@astanait.edu.kz

Main part:

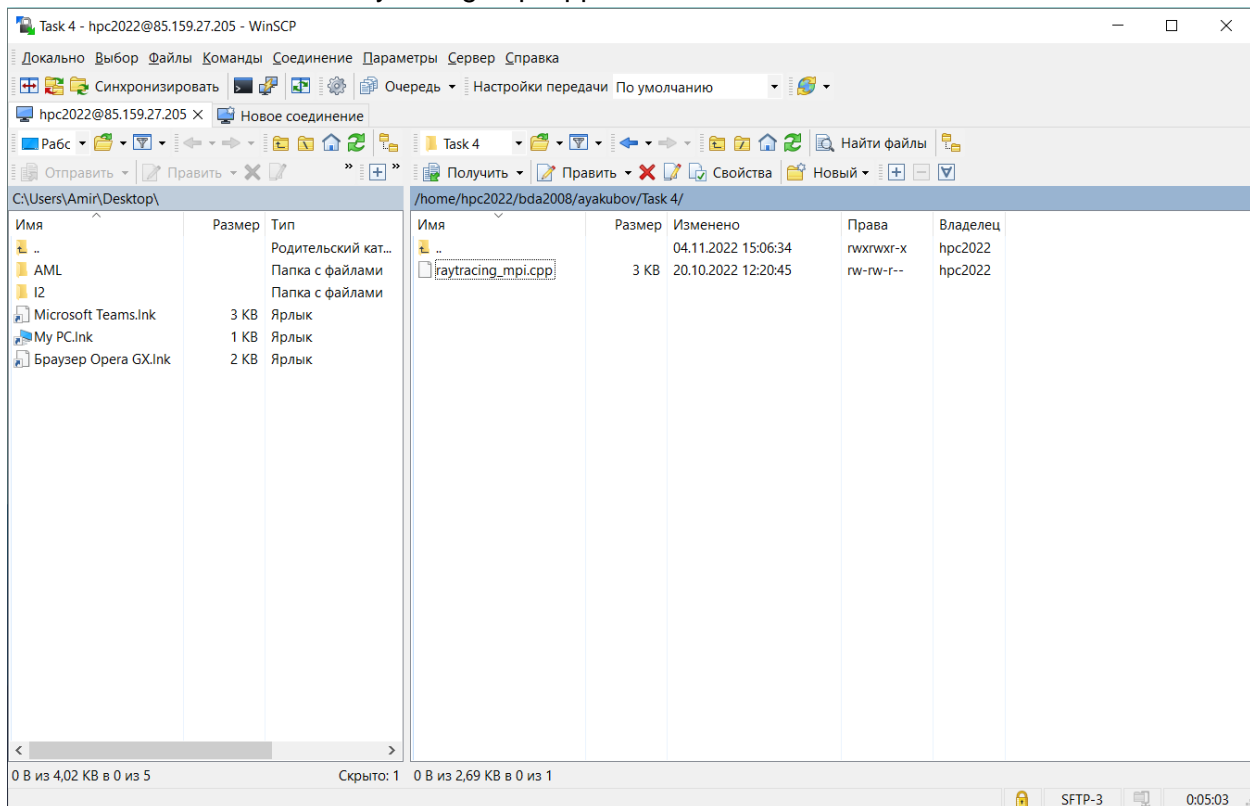
Step 0:

We will be using the sequential ray tracing program from Task 1. Download and install Mini-Rt library (<https://github.com/georgy-schukin/mini-rt>), if necessary.

Step 1: Prepare a directory for the Task 4

In your personal directory:

- Create directory "Task 4"
- Copy sequential program to this new directory
- Rename the file to raytracing_mpi.cpp



Step 2: Implement parallel program with MPI

Use MPI to parallelize the sequential ray tracing program (edit `raytracing_mpi.cpp`). In your parallel program:

1. Image to compute should be partitioned into blocks
2. Image blocks should be distributed among processes, processes should compute their block(s) in parallel
3. The main process (for example, rank 0) should receive computed blocks from all processes and create the final image file

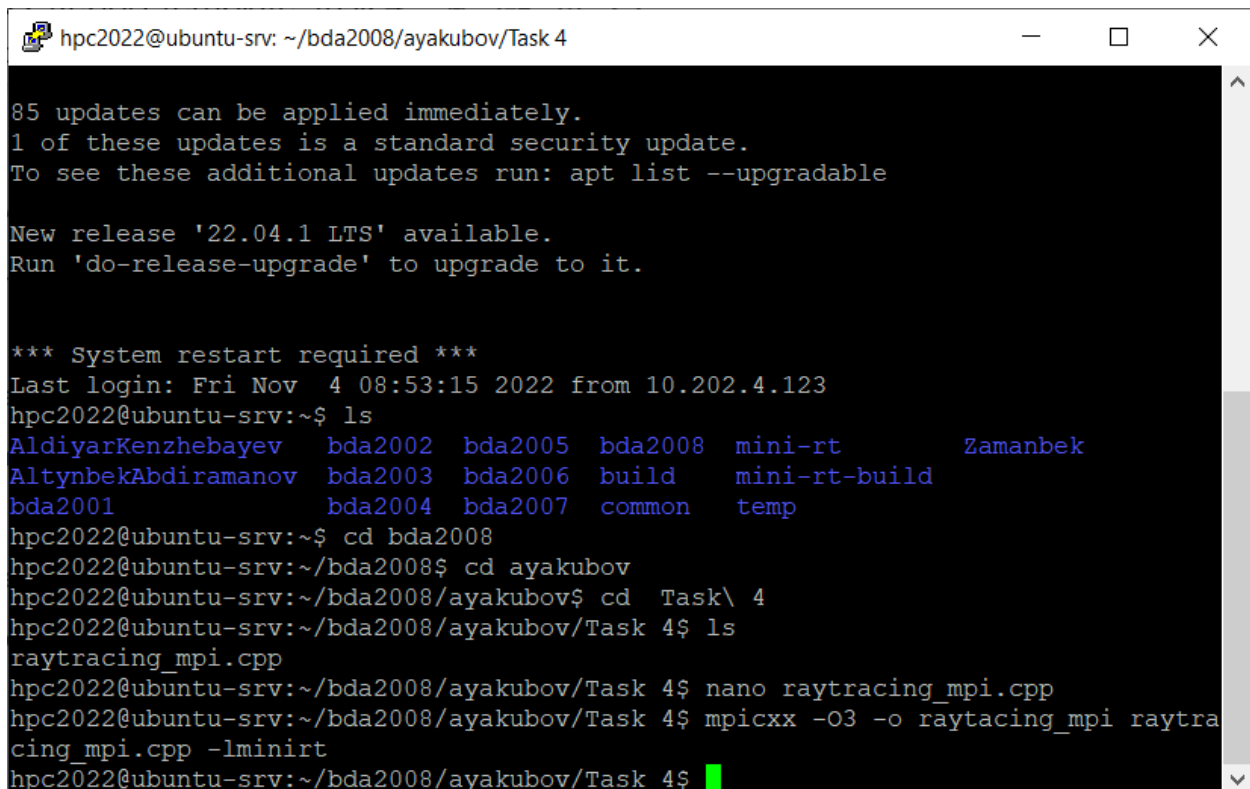
You can implement any way of partitioning the image onto blocks and distributing blocks (a single block for each process or multiple blocks for each process, the same size for all blocks or different block sizes, static distribution of blocks or dynamic distribution, etc).

Hint: you can use [this program template](#) as a starting point.

Hint: study [this program example](#) about parallelizing computation of an array with MPI.

To compile MPI program:

```
mpicxx -O3 -o raytacing_mpi raytracing_mpi.cpp -lminirt
```



```
hpc2022@ubuntu-srv: ~/bda2008/ayakubov/Task 4

85 updates can be applied immediately.
1 of these updates is a standard security update.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Fri Nov  4 08:53:15 2022 from 10.202.4.123
hpc2022@ubuntu-srv:~$ ls
AldiyarKenzhebayev  bda2002  bda2005  bda2008  mini-rt      Zamanbek
AltynbekAbdiramanov bda2003  bda2006  build    mini-rt-build
bda2001             bda2004  bda2007  common   temp
hpc2022@ubuntu-srv:~$ cd bda2008
hpc2022@ubuntu-srv:~/bda2008$ cd ayakubov
hpc2022@ubuntu-srv:~/bda2008/ayakubov$ cd Task\ 4
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ ls
raytracing_mpi.cpp
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ nano raytracing_mpi.cpp
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpicxx -O3 -o raytacing_mpi raytracing_mpi.cpp -lminirt
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$
```

```
hpc2022@ubuntu-srv: ~/bda2008/ayakubov/Task 4
Swap usage: 3%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment.

  https://ubuntu.com/engage/secure-kubernetes-at-the-edge

85 updates can be applied immediately.
1 of these updates is a standard security update.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Fri Nov  4 09:57:29 2022 from 10.202.18.115
hpc2022@ubuntu-srv:~$ cd bda2008
hpc2022@ubuntu-srv:~/bda2008$ cd ayakubov
hpc2022@ubuntu-srv:~/bda2008/ayakubov$ cd Task\ 4
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ nano raytracing_mpi.cpp
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpicxx -O3 -o raytracing_mpi raytr
acing_mpi.cpp -lminirt
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$
```

To run MPI program with N processes:

```
mpiexec -np N ./raytracing_mpi <args>
```

Step 3: Study performance of your parallel program

1. Use `MPI_Wtime()` to measure the execution time for the parallelized main loop on each process (the time to compute all image block(s) for a process), then compute the final execution time for the program (max time among all processes):

```
double start = MPI_Wtime();
    for(int x = ...)
        for(int y = ...) {
            const auto color = viewPlane.computePixel(
                scene, x, y, numOfSamples);
            ...
        }
double end = MPI_Wtime();

double execution_time = end - start;

// Choose the maximal execution_time among all the processes
//(use MPI_Reduce for that)
```

```
// Output the maximal execution_time
if(rank == 0) {
    std::cout << "time = " << max_execution_time << std::endl;
}
```

2. Select such a scene and rendering parameters (image size, number of samples, depth of recursion, etc.), that the execution time of the program, when running on 1 process, is more than several seconds.
3. Measure the execution time for the parallel program on 1, 2, 4, 8, 16 processes. For accuracy you can do several runs (>5) on each number of processes and choose the minimal time among runs for this number of processes.

```
hpc2022@ubuntu-srv: ~/bda2008/ayakubov/Task 4
New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Fri Nov  4 10:21:17 2022 from 10.202.21.139
hpc2022@ubuntu-srv:~$ cd bda2008
hpc2022@ubuntu-srv:~/bda2008$ cd ayakubov
hpc2022@ubuntu-srv:~/bda2008/ayakubov$ cd Task\ 4
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpiexec -np 1 ./raytracing_mpi 640 480 10
Time = 7.64677
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpiexec -np 2 ./raytracing_mpi 640 480 10
Time = 3.62696
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpiexec -np 4 ./raytracing_mpi 640 480 10
Time = 3.09654
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpiexec -np 8 ./raytracing_mpi 640 480 10
Time = 2.21411
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$ mpiexec -np 16 ./raytracing_mpi 640 480 10
Time = 1.90851
hpc2022@ubuntu-srv:~/bda2008/ayakubov/Task 4$
```

4. Build plots/tables for:
 - a. The execution time (to demonstrate how it depends on the number of processes)
 - b. Speedup: $\text{Speedup}(N) = \text{Time}(1) / \text{Time}(N)$, N - number of processes
 - c. Efficiency: $\text{Efficiency}(N) = \text{Speedup}(N) / N$

Number of threads	Execution time	Speedup(N)	Efficiency
1	7.64677	1	1
2	3.62696	2.10831	0.95485
4	3.09654	2.46945	0.61736
8	2.21411	3.44365	0.43045
16	1.90851	4.00667	0.25041

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

Upload your source code files in Task 4 directory to your Github repository, include a link to it in the report.

Link: <https://github.com/Am1rrr/hpc/tree/main/Task%204>

Step 5: Conclusion in a free form

For shared memory (using OpenMP) and distributed memory (using MPI), we have developed an object-oriented parallel raytracer. While the performance measurement is being done, the load calculation option can be disabled and then reactivated at build time. Experimental findings demonstrate that in both MPI and OpenMP instances, the load balancing technique used delivers speedup and close to ideal efficiency, with even better performance in the event of unequal object distribution.