Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing **2022**

Student: SIMONE TARENZI Discussed with: FULL NAME

Solution for Project 5

Due date: 7.12.2022, 23:59

1. Task 1 - Initialize and finalize MPI [5 Points]

Initialized MPI with MPI_Init_thread by passing argc, argv, and MPI_THREAD_FUNNELED, got the rank with MPI_Comm_rank and the size with MPI_Comm_size, and finalized with MPI_Finalize.

2. Task 2 - Create a Cartesian topology [10 Points]

First I used MPI_Dims_create to create the number of partitions depending on mpi_size, then I used MPI_Cart_create to create the communicator which will be used to communicate with the neighbouring topology. After that I used MPI_Cart_coords to retrieve the coordinates of each process, and finally MPI_Cart_shift to get the neighbours of each rank.

3. Task 3 - Change linear algebra functions [5 Points]

I used MPI_Allreduce to get the reduced result of the two operations. It's only needed in those because the two functions are working on the same result between all processes, and so there must be no race conditions to get it correct.

4. Task 4 - Exchange ghost cells [45 Points]

Thanks to the given code for receiving and sending data to the northern neighbour, implementing the rest of the code for the other neighbours was very straighforward.
I put MPI_Waitall after the interior grid point are calculated since they are not sent to the neighbours.

5. Task 5 - Testing [20 Points]

I used a simple bash script to iterate the various testing required, from 1 to 10 OMP threads.
The results were saved in their own .txt file depending on the matrix size and on the number of ranks used.
After that, I used python to plot the results into a graph for each grid size. (I'm sorry in advance for the graphs, they are not very good. The more precise results can be found in the .txt files.)

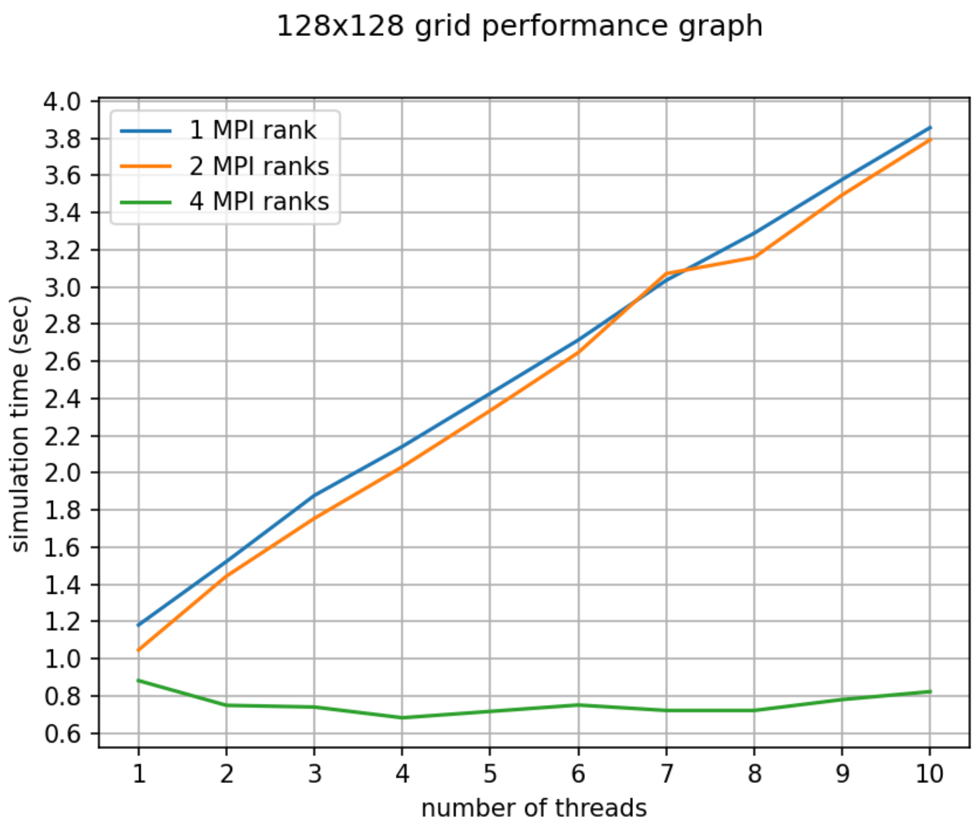

Figure 1: performance graph of the testing done on a 128 by 128 matrix

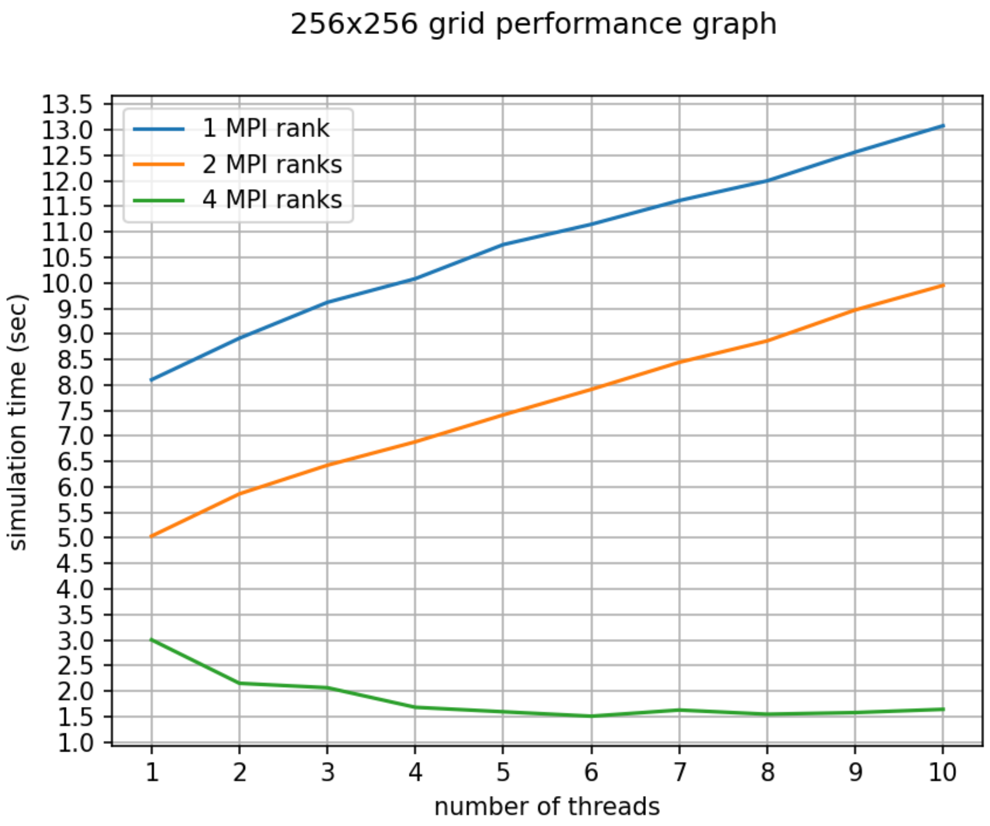

Figure 2: performance graph of the testing done on a 256 by 256 matrix

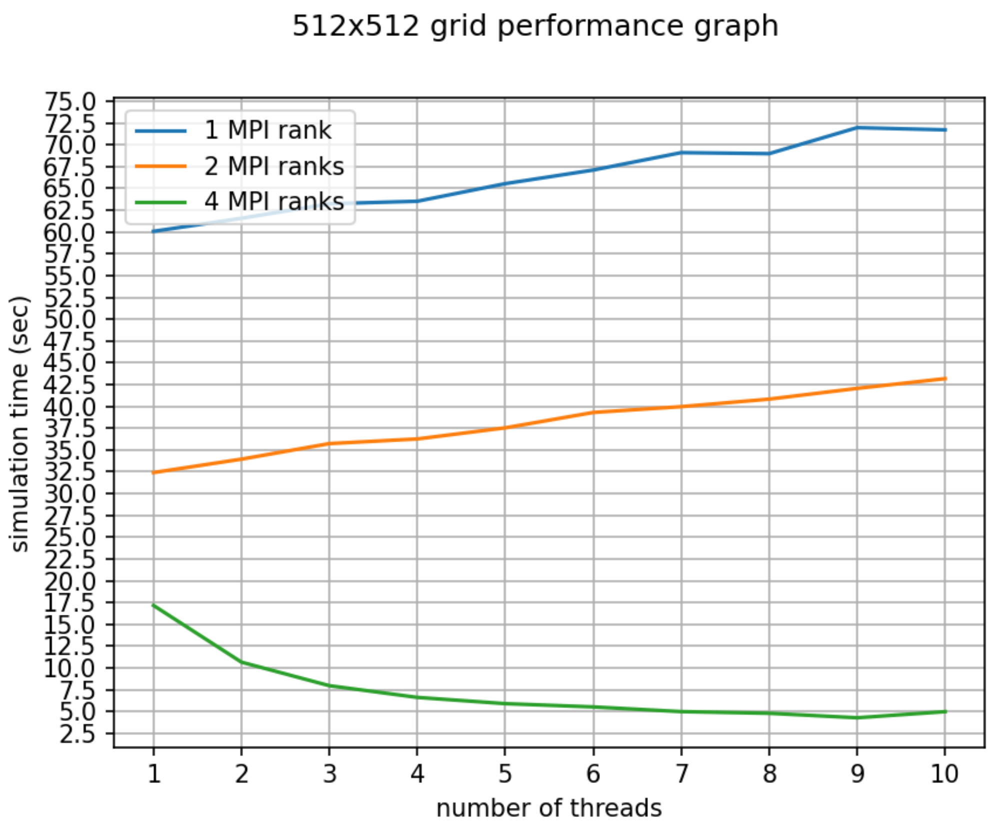

Figure 3: performance graph of the testing done on a 512 by 512 matrix

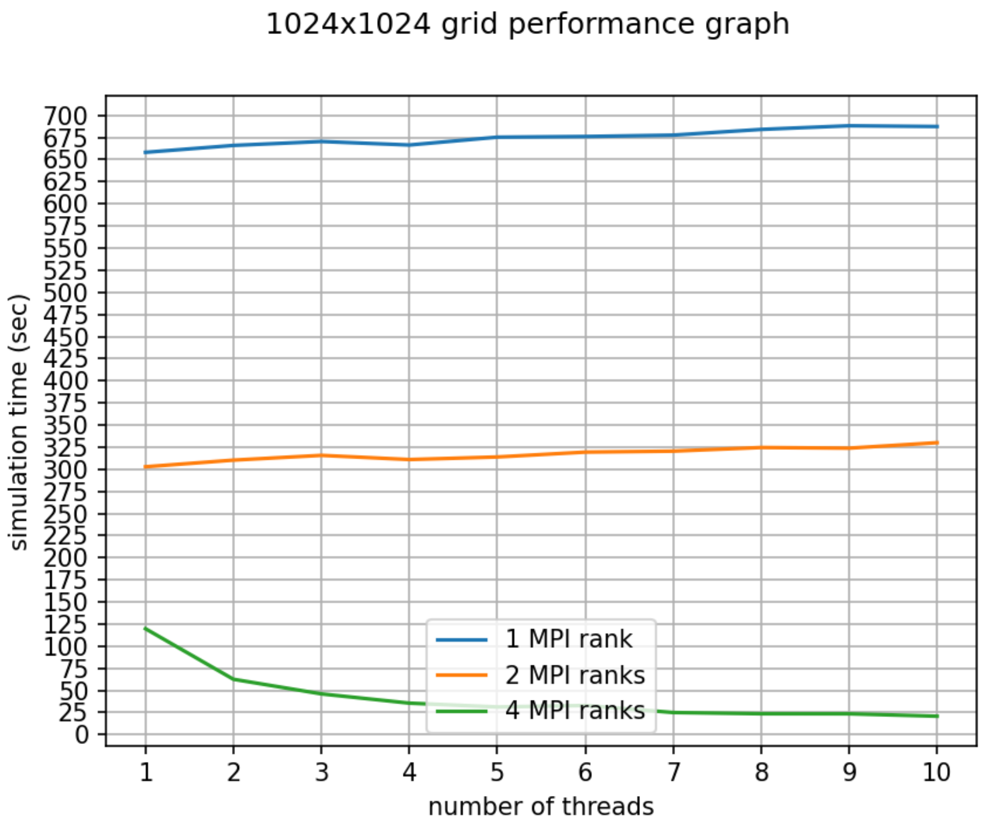

Figure 4: performance graph of the testing done on a 1024 by 1024 matrix

Some observations:

- Increasing the number of MPI ranks always improves performance: The bigger the matrix, the bigger the difference between them. Starting from the 256 by 256 matrix, doubling the amount of MPI ranks also doubles the performance.
- Increasing the number of threads always worsens performance when using 1 or 2 MPI ranks. However, the loss in performance gets smaller as the matrix size increases: it's possible that with higher matrices than the ones tested, performance could improve with more threads.
- Increasing the number of threads improves performance when using 4 MPI ranks: in the 128 by 128 matrix the best results were had when using 4 threads, in the 256 by 256 one when using 6, in the 512 by 512 when using 9, and in the 1024 by 1024 when using 10 (the increase in performance also looks logarithmic starting from the 256 by 256 matrix).

6. Task 6 - Quality of the Report [15 Points]