Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing **2022**

Student: CHUKWUNONYE AGBAKURU-NWOGU Discussed with: FULL NULL

Solution for Project 5

Due date: 30.11.2022, 23:59

HPC 2022 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

This project will introduce you a parallel space solution of a nonlinear PDE using MPI.

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

In the main.cpp file, the functions below were make for specific purposes
'MPI Init' initializes MPI.
'MPI Comm rank' retrieves the current rank.
'MPI Comm size' gets the number of ranks.
'MPI Finalize' is used to finalizes the MPI.

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

In the data.cpp file, the functions below were made in order to perform specific functionalities.
'MPI Dims create' is responsible for creating a certain number of subdomains.
'MPI Cart create' creates a Cartesian communicator.
'MPI Cart coords' defines the coordinates of a specific rank in the Cartesian topology.
'MPI Cart shift' determines the neighbours of a specific rank.

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

In the linalg.cpp file, MPI Allreduce is a collective operation which is initiated to implement the dot product, and the norm. It enables the local computation of the dot product and allows it to be shared and distributed to different ranks.
The vector needs to be distributed over all the ranks because all the entries need to communicate as the entries of the input vector are used in order to create the vector that would be outputted.

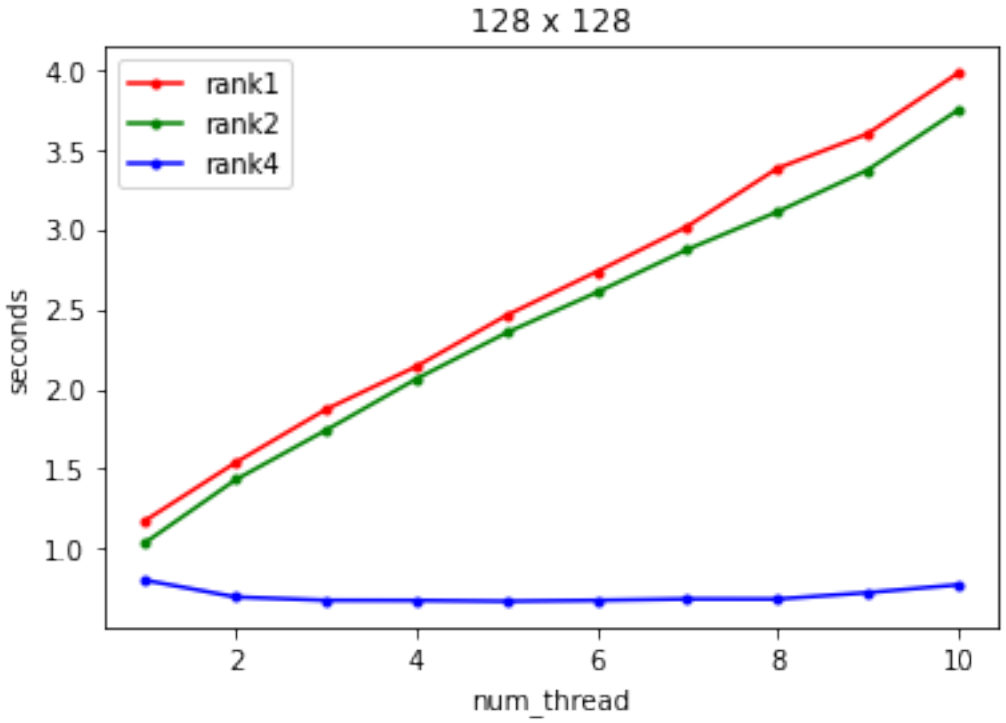
4. Task 4 - Exchange ghost cells [45 Points]

In the operators.cpp file, values are sent to the buffers (buffN, buffS, buffE, buffW), and received by the ghost cells (bndN, nbdS, bndE, bndW). This is how the ghost cells are exchanged with neighbors. They are sent by using the MPI Isend function, and received by using the MPI Irecv function. The MPI Waitall function waits for the communication to finish. This is how the non-blocking communication is implemented.

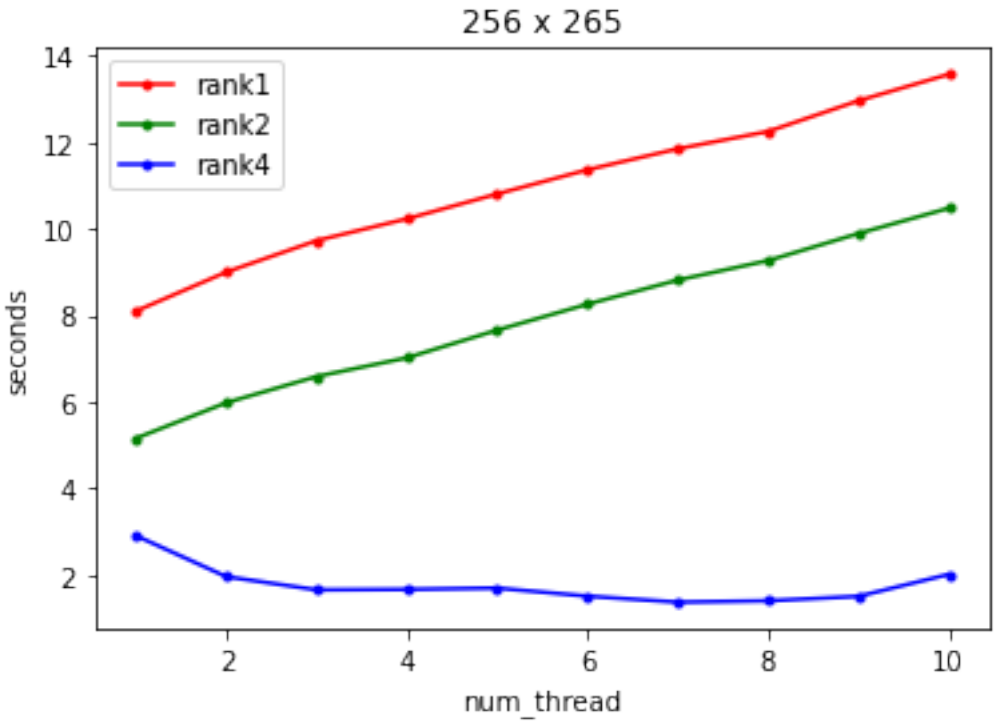
5. Task 5 - Testing [20 Points]

Strong scaling was performed in order to test how it scales at different resolutions.
The resolution sizes that were tested were:
128 x 128, 256 x 256, 512 x 512 and 1024 x 1024.

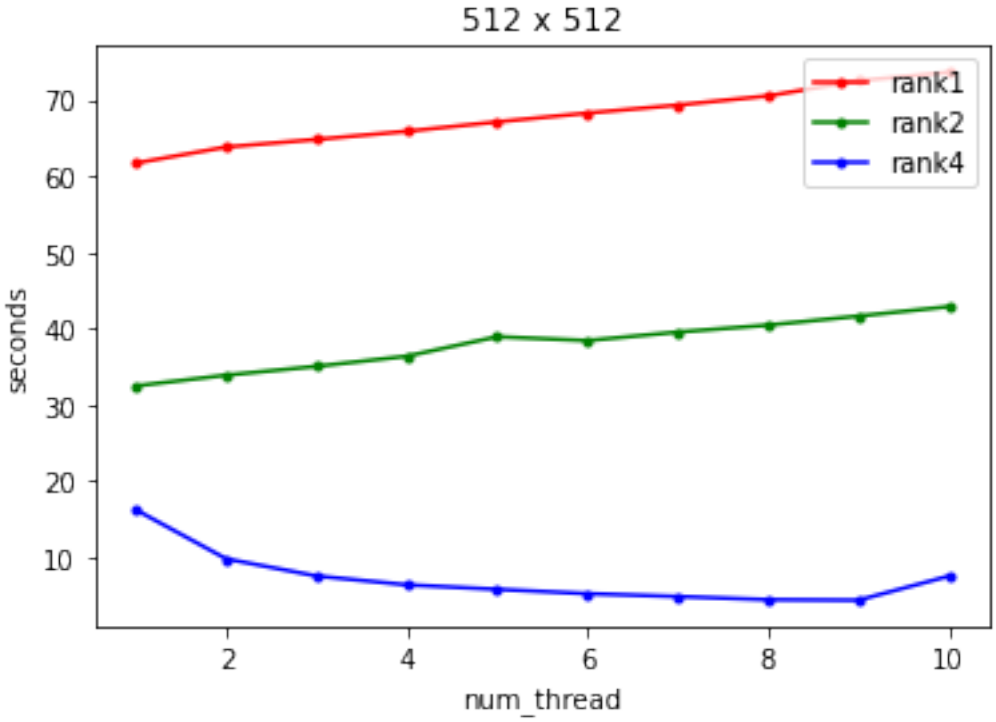
These grid sizes were tested with 1-10 threads, and 1, 2, 4 MPI ranks.
Each image shows the speed at which each grid was computed, at different MPI ranks.



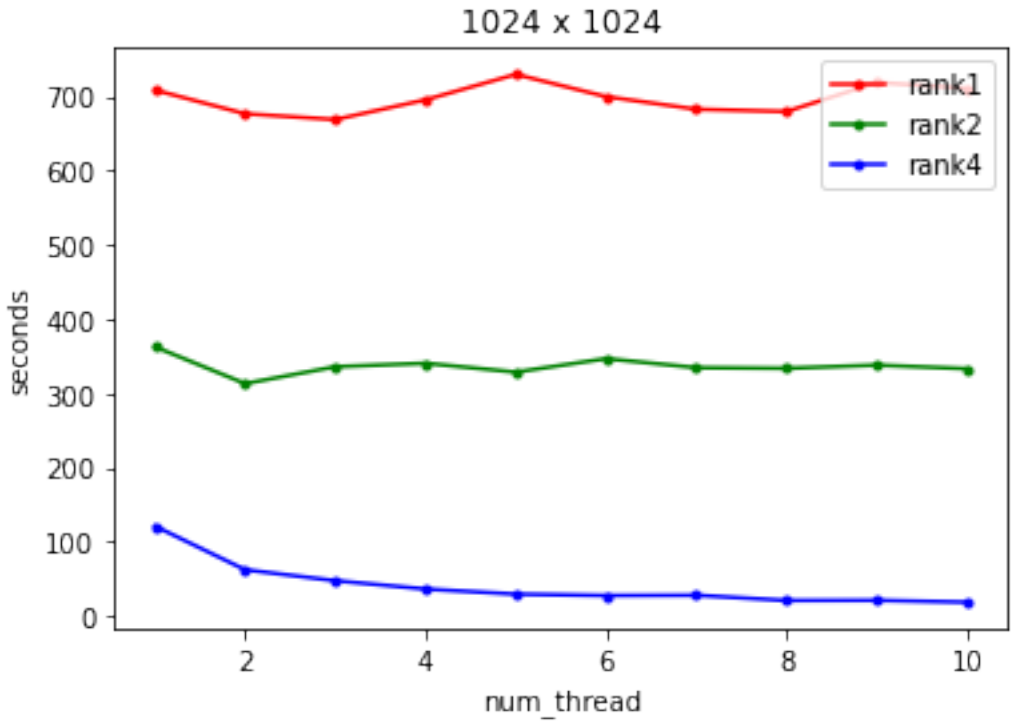
(Fig 1.: 128 x 128)



(Fig 2.: 256 x 256)



(Fig 3.: 512 x 512)



(Fig 4.: 1024 x 1024)

Each process within a communicator has a unique rank. Saying 'MPI rank 4' means the process ranked 4th in a communicator.
Looking at the images above, it is clear that the 4th process outperformed the 2nd and 1st. This is because it computed the grids in less time than the others. The speed of the 4th ranked process comes at a cost of processing power. It would require a lot of processing power and more temporary memory in order to compute the grids with such speed. With this large consumption of temporary space required to perform rapid computations, it could result in cache-misses.

Although the 4th process rank improved the speed of computation, the amount of threads seem to have no effects on the computation time. The more the threads, the faster the computation should be. This is because threads enable parallel processing where multiple threads are able to compute a single task.
Strangely, as the amount of threads increased, the amount of seconds used for computation also increased in grid sizes 128 and 256. This is strange because the computation should have decreased instead of increased.

In my opinion, I would say that this scales well because although the performance of the threads are not very evident, the MPI ranks still did a good job in decreasing the computation time. It is evident in all the graphs for the grids that the computation time reduced because of the implementation of MPI rank 4.

Weak scaling could not be performed because no more than 4 ranks could be used. If it were to be performed,
grid size 128 would be tested with 16 ranks
grid size 265 would be tested with 32 ranks
grid size 512 would be tested with 64 ranks
and grid size would be tested with 128 ranks.

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

Additional notes and submission details

Submit the source code files (together with your used `Makefile`) in an archive file (tar, zip, etc.) and summarize your results and the observations for all exercises by writing an extended Latex report. Use the Latex template from the webpage and upload the Latex summary as a PDF to iCorsi.

- Your submission should be a gzipped tar archive, formatted like project_number_lastname_firstname.zip or project_number_lastname_firstname.tgz. It should contain:
  - all the source codes of your MPI solutions.
  - your write-up with your name project_number_lastname_firstname.pdf,
- Submit your .tgz through Icorsi.