Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing **2022**

Student: Nicolai Hermann Discussed with: Michele Cattaneo

Solution for Project 4

Due date: 23.11.2022, 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.

1. Ring maximum using MPI [10 Points]

To determine the right neighbour of some rank, +1 is simply added to the rank. However for the rightmost rank the right neighbour will be the leftmost neighbour. Equivalently for the left neighbour, the left neighbour of the first rank is the rightmost neighbour (size-1).

For the ring addition the send buffer is initialized with `my_rank`. Using `MPI_Sendrecv` the send buffer is sent to the right neighbour and simultaneously a message from the left neighbour is received. The received massage is then added to the sum and copied to the send buffer to pass it on to the right neighbour in the next iteration. The output aligns with the output shown in the assignment.

2. Ghost cells exchange between neighboring processes [15 Points]

In order to create the Cartesian communicator the `MPI_Cart_create` function was used. Afterwards the ranks of the neighbours were inferred using the `MPI_Cart_shift`, by displacing the Cartesian topology in units of one in the respective dimensions to get the right/left or top/bottom neighbours. The `MPI_Type_vector` was utilized for the column ghost cells. A starting point will be specified later and by skipping `DOMAINSIZE` many doubles we can iterate through the column. To send the

correct data to the respective neighbour it was only needed to send the pointer of the first respective element. For the left and right neighbour the vector was used since a column should be sent. The tag was a range from 0-3 specifying the direction starting from top and moving clockwise. Therefore sending to the top neighbour had the 0 tag. However to receive the top ghost cells we actually want to receive data with a bottom tag as the current cell will be the top neighbour bottom. The same principal continues for the remaining constellations. Finally the expected result was obtained an is depicted in the following Fig. 1.

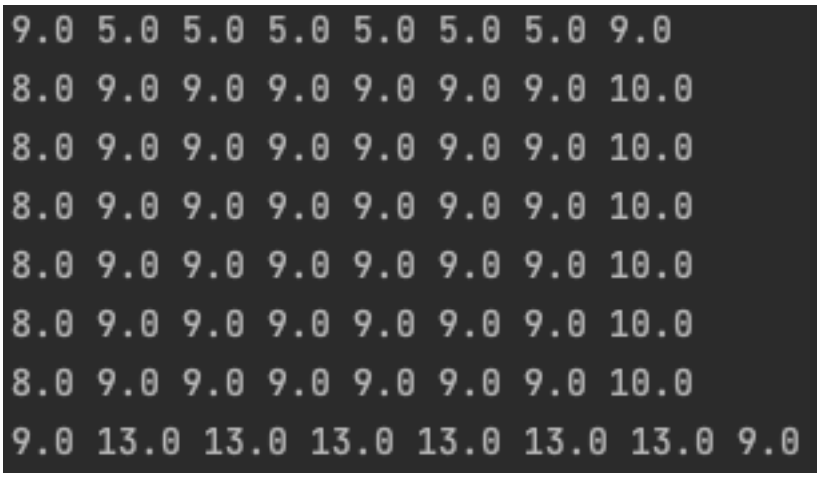

Figure 1: Output of the ghost cell exchange between neighboring processes for process number (rank) 9.

3. Parallelizing the Mandelbrot set using MPI [20 Points]

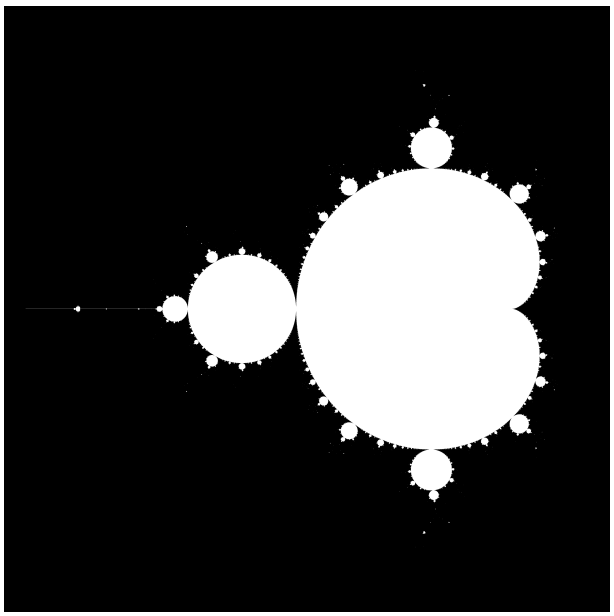

Figure 2: Mandelbrot result with 16 open mpi processes.

The `createPartition` function was straight forward to implement when using the functions specified in the todo comments. The Cartesian communicator is not periodic. Same thing for the `updatePartition`, the functions specified were used.

For the `createDomain` to get the `nx` for the domain, the respective dimension (width or height for `ny`) was divided by the `nx`/`ny` of the partition. After knowing how many pixels the domain has,

the start was computed by multiplying that amount by the x/y coordinate of the process in the Cartesian grid. The end was simply `d.nx` plus the start minus one.

To transmit the data to the master process (0) a blocking send and receive was used in the respective processes. The results of performance experiment can be seen in Fig. 3. The graph depicts how long each process took for different number of processes. It becomes immediately apparent, that the work load was not distributed equally. In this case all bar would have the same height. The result however is coherent as the convergence and thus computation for white pixels takes a lot longer than for black pixels. Since the image is split into chunks, some processes compute mostly black pixels and a few mostly white pixels leading to the measured work load imbalance.

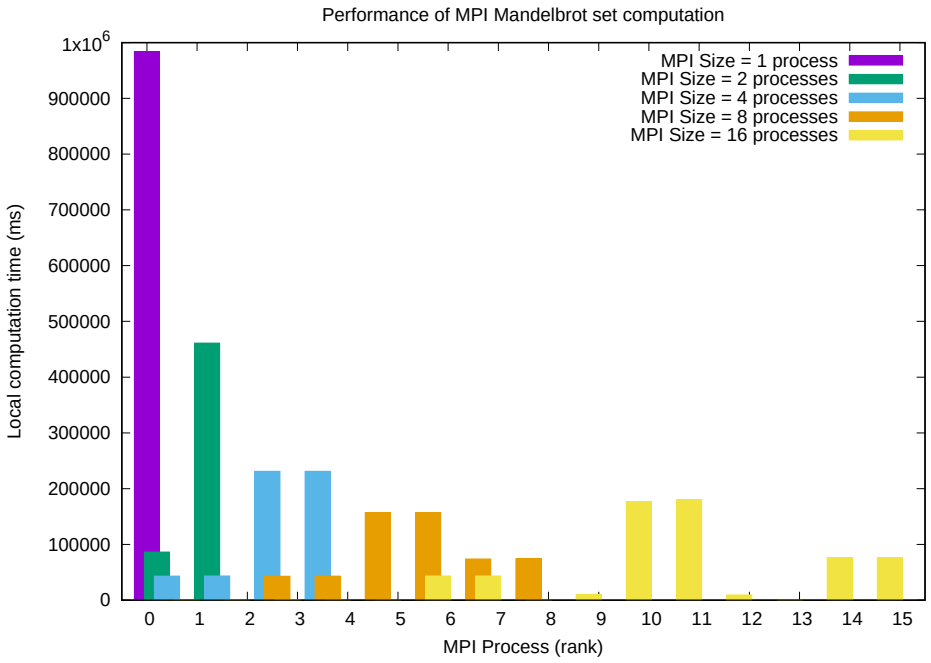

Figure 3: Computation time of each process with varying numbers of processes. The group of processes belonging to the same run are depicted in the same color.

4. Option A: Parallel matrix-vector multiplication and the power method [40 Points]

To start off, a pointer to the vector x was allocated for all processes and the number of rows per process was calculated. The respective part of the matrix A was generated for each process. The master process initializes the vector x with random values. Afterwards the timer starts and the powermethod function is called. First of, the master thread normalizes x and broadcasts the resulting vector to all processes with `MPI_Bcast`. With the updated x the `matVec` function is called with x and the partial matrix A to obtain a new partial vector y. Finally all the partial ys have to be accumulated in the master thread which was accomplished with the `MPI_Gather` function. It takes all partial ys and writes them back to x in the order of the rank number. Now the for loop repeats and the master thread normalizes the new x, ... After a certain amount of iterations the powermethod returns and the norm of x resembles the norm of the dominant eigenvalue of A. The master process stops the timer and the validate method is called.

4.1. Scaling Analysis

Unfortunately the cluster was immensely occupied with approximately 3400 pending job submissions on Wednesday making it almost impossible to work or run benchmarks/ experiments. It took more than an hour to allocate 16+ nodes which strongly slowed down any progress making this assignment very frustrating. The result of `squeue` is submitted alongside the code in `queue.txt` to prove the occupation.

Even though I was not able to run the data collection I still want to theorize how the plots probably would have looked like. For the strong scaling analysis the increase in processes will strongly benefit the overall performance. The larger n the stronger will be the effect as there is more work to distribute. If n is too small the performance will start to decline again as the communication between processes over weighs the actual task. Especially if more processes are used than available on one ICS cluster node. This will force the program to use more than one ICS cluster node which is even more expensive. The parallel efficiency will be great at the beginning and as soon as the either the problem split gets too small or another ICS cluster node has to be used the efficiency will drop.

The weak scaling analysis would show a shallow linear increase in computation time. Most likely the slope will not drastically grow as the we go up to 64 processes. However as soon as multiple ICS cluster nodes are used the slope will increase a good amount as the messaging latency will increase. The parallel efficiency should decline rather quickly compared to the linear increase of the weak scaling. The computation time increases for every new process and since the denominator of the parallel efficiency formula is $n \cdot T_n$ where n is the number of processes and T_n the execution time using n processes, the denominator will get larger quickly thus the parallel efficiency smaller/ worse.

5. Option B: Parallel PageRank Algorithm and the Power method [40 Points]

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

Each project will have 100 points (out of which 15 points will be given to the general quality of the written report).

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.