ETH zürich

High-Performance Computing Lab for CSE **2021**

Student: Sina Klampt Discussed with: Alain Hügli, Anna Hutter

Solution for Project 4

Due date: 26 April 2021 (midnight)

HPC Lab for CSE 2021 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to Moodle (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]

In this task, I calculated the right neighbors using "rank + 1 modulo size" while using "rank - 1 modulo size" for the left neighbors. With this implementation the ring functionality was given. Additionally I used a do-while loop with a "Sendrecv" routine which sent a send buffer and received the result in a receive buffer. In the end I stored the receive buffer in the send buffer.

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

First I set the dimensions and periods to the right values and used a "Cart create" function to create a communicator. Then I found the correct neighbors using the "Cart shift" function. On my first attempt I swapped the two coordinate directions and thus got the "transposed" output compared to the one we were supposed to get. However once I swapped the two, I got the right results.

I then committed a new vector with stride 8. For the sending and receiving it was a bit difficult to figure out the correct order and the right initial addresses.

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

First we needed to implement the functions in the "consts.h" file. In the "createPartition" function I used MPI Dims create, MPI Cart create and MPI Cart coords for the grid size, the cartesian

communicator and the coordinates respectively. I used MPI Cart coords again in the "updatePartition" function to update the coordinates in the grid. Lastly in the "createDomain" function we had to add the size, the index of the first pixel and the index of the last pixel in the local domain.

In the "mandel_mpi.c" file we needed to send the local partition to the master process and receive the partition of the process into an array.

In the figures below you can see the performance (Fig. 1) and a comparison of the images using 1 or 16 processors (Fig. 2). We can see that the speedup is quite good, since the performance with 4 processors is takes only half the time compared to using 1 processor. However we could not get better results with 16 processors compared to 8 processors. To further improve the performance one could maybe partition the workload better between the different processors.

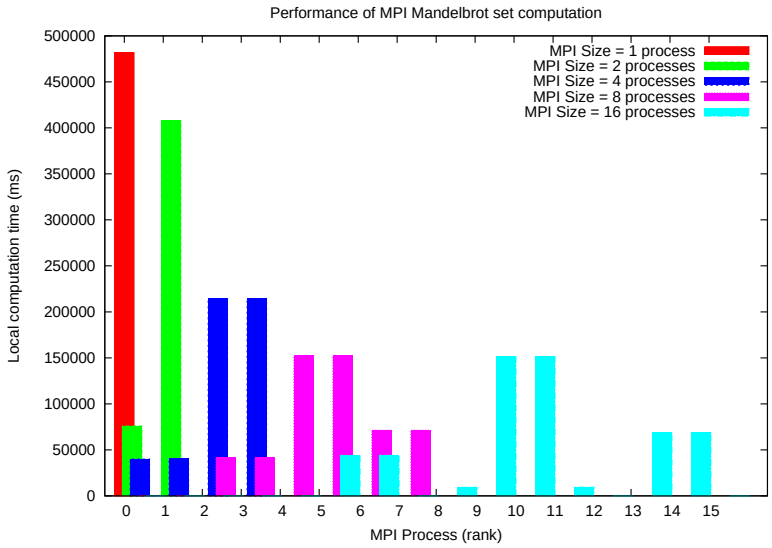
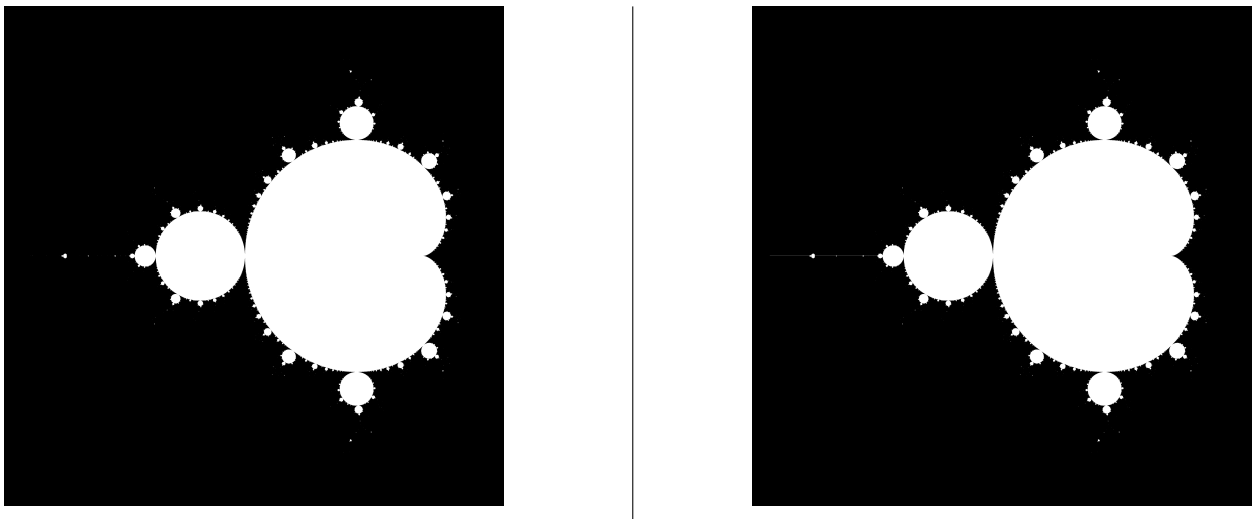


Figure 1: Performance



(a) 1 processor

(b) 16 processors

Figure 2: Slight difference in the two plots. There is a fine line on the left hand side of plot b) which is missing in a).

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

In the "powermethod.c" file we had to implement different functions as stated in the task. In the main function I generated a matrix and called the function "powermethod". To stop the time I additionally called the "hpc_timer" function right before and after.

The implementation of the "norm" function was pretty straight forward as we have done these kinds of implementations countless times before and did not have to use any MPI functions. To implement the "powerMethod" function, I initialized a vector with size N with random values. Then I created a loop where the vector is normalized and calls the function "matVec". The last function to implement was the "matVec" function where we implemented the parallelization. I then broadcasted the vector to all processes using MPI Bcast. I allocated a vector on each process and computed the matrix vector multiplication. The result was then stored in the local vector. Finally I gathered all the local vectors into a final result vector using MPI Gather.

You can find my results for the strong scaling plot (Fig. 3) and the weak scaling plot (Fig. 4) below. We can see in the strong scaling plot that the time cuts in half when the number of processors is doubled.

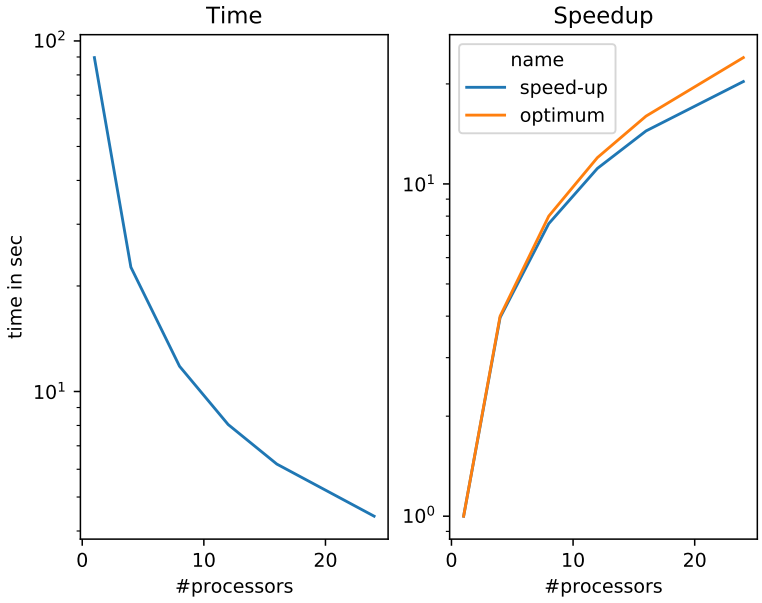


Figure 3: Strong scaling

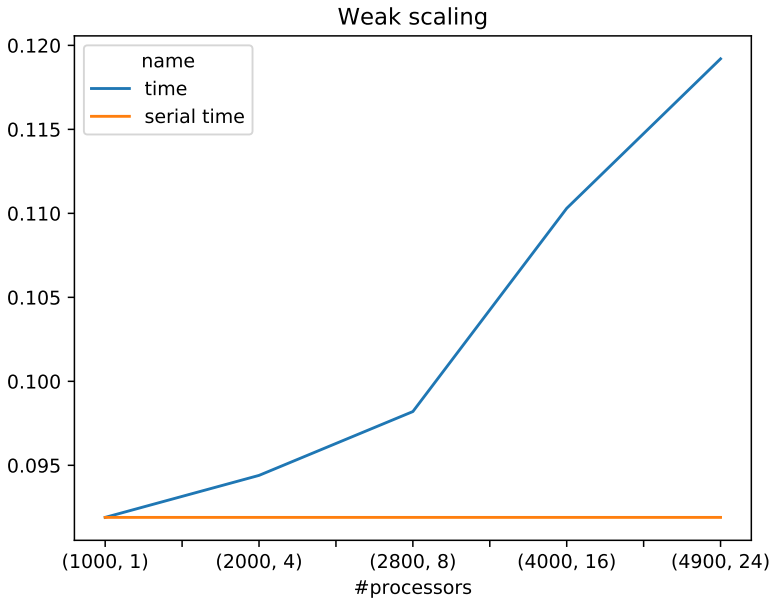


Figure 4: Weak scaling

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 Moodle.

- 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 .zip/.tgz through Moodle.