ETH zürich

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

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

Solution for Project 2

Due date: 26.03.2021 (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.

This project will introduce you to parallel programming using OpenMP.

1. Parallel reduction operations using OpenMP [10 points]

In this part we had to implement two different approaches. The reduction part was easily implemented with one line of code. For the critical version I used a thread local variable to do parallelized for-loop computations and aggregated the results into a variable that stored the final result in the critical section.

You can see my results in the following 3 plots:

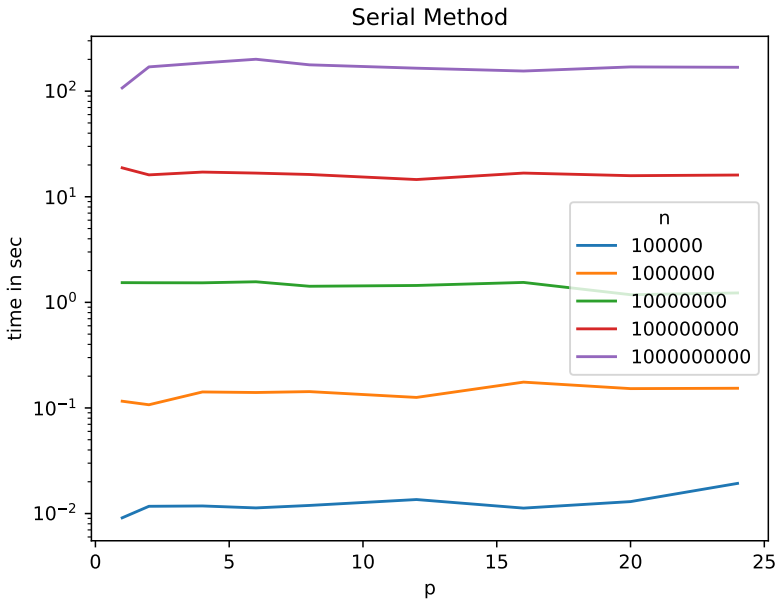

Figure 1: Perfect Scaling Serial

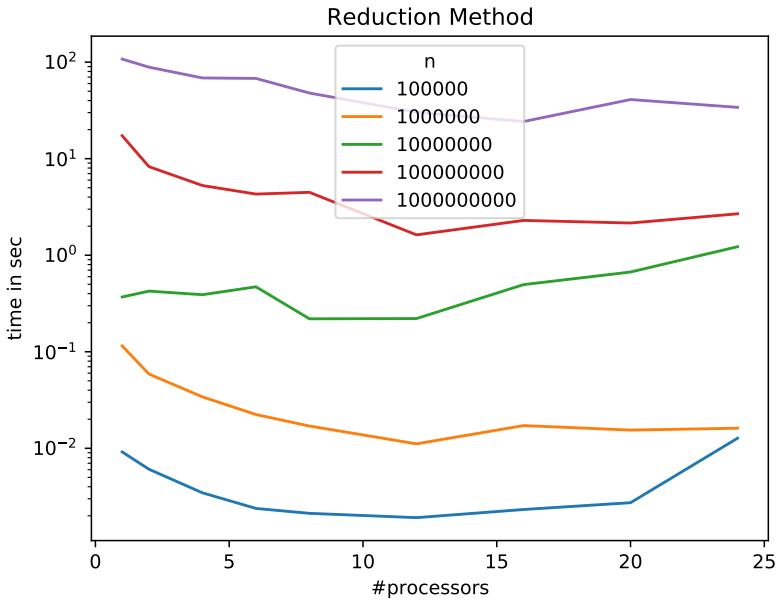

Figure 2: Perfect Scaling Reduction

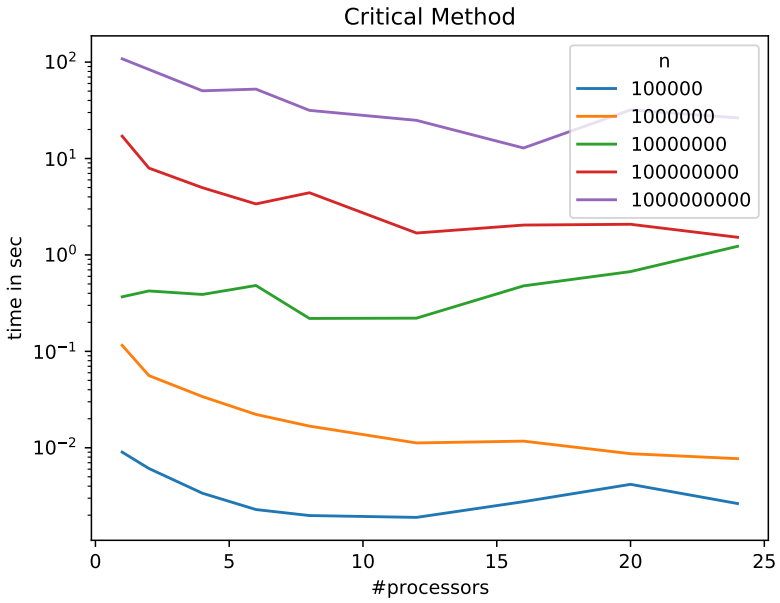

Figure 3: Perfect Scaling Critical

2. The Mandelbrot set using OpenMP [30 points]

For the first and second part of the exercise I was able to follow the instructions given to implement the corresponding code. The tricky part was to figure out which variable to update first. I updated y first because its value depends on x.

Unfortunately I did not have time to fully do the third part of the exercise since there were some issues with running the code on the euler cluster. Somehow my makefile did not compile.

3. Bug hunt [15 points]

3.1.

The bug of the first code is that "tid" in line 26 should be inside the following for-loop.

3.2.

For the second bug the variables "tid" and "i" should be private since every thread has its own id or is the iterator respectively.

3.3.

The third bug is the barrier in line 79. Barriers can only be used when we are sure that every thread is going to call it which is not the case here. Using this inside the "print results" function results in the threads doing the work in the sections never finish. In this case we should remove the barrier.

3.4.

The forth bug is that the matrix is too large. We can resolve this issue by increasing the stack size.

3.5.

In the fifth bug we have deadlocks. In the first section "a" gets locked first and then "b", in the other section it is the other way around. To resolve this issue we should unlock first before we lock again.

4. Parallel histogram calculation using OpenMP [15 points]

For my initial approach I initialized a new thread local array (length = number of bins). The for-loop over the vecsize could then be done in parallel. Then I looped over the number of bins in a critical section to aggregate the results that were calculated before. I think this works well since the number of bins is small in comparison to the vecsize.

You can see my results in the following table:

Sequential	1.189 s
1 Thread	1.218 s
4 Threads	0.626 s
8 Threads	0.425 s
16 Threads	0.303 s

5. Parallel loop dependencies with OpenMP [15 points]

I have never heard of the first/last private before and learned that with static scheduling the workload assigned to a thread are consecutive loop iterations. First I tried to implement it with the pow function, but because this is a rather expensive operation I didn't get a good performance for the calculation of the appropriate values of Sn. In another attempt I initialized a thread local flag that shows if an iteration is the first of a given thread. With this method I only had to compute pow once (for the first iteration of every thread) and was then able to multiply by "up" to get the values for Sn. I set Sn to be last private since we are interested in the last Sn. I also the flag to private because every thread needs to modify it.

You can see my results in the following table:

Sequential	5.012 s
1 Thread	8.845 s
4 Threads	2.153 s
8 Threads	1.283 s
16 Threads	1.946 s

6. Task: 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 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 OpenMP solutions.
  - your write-up with your name project_number_lastname_firstname.pdf,
- Submit your .tgz through Moodle.