ETH zürich

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

Student: Sina Klampt Discussed with: Alain Hügli

Solution for Project 3

Due date: 12 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.

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

1. Task: Implementing the linear algebra functions and the stencil operators [35 Points]

In the first part we had to implement the linear algebra functions. This task was pretty straightforward since we only had to implement what was already stated in the respective comments.

In the second part we needed to implement the stencil kernels. After some initial problems figuring out the right indices and entries that were null and therefore omittable, I was able to implement it correctly and get the same results as described in the task.

You can see my result in the following plot:

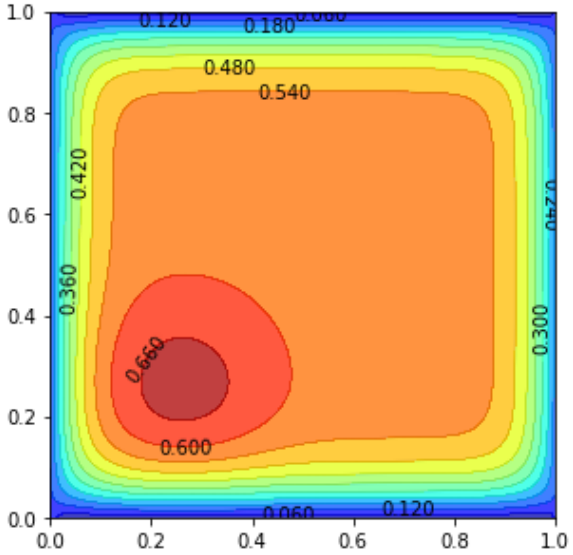

Figure 1: Output of the mini-app for a domain discretization into 128x128 grid points, 100 time steps with a simulation time from 0 - 0.005s

2. Task: Adding OpenMP to the nonlinear PDE mini-app [50 Points]

Replace welcome message in main.cpp [2 Points]

In this part we had to replace the welcome message. The first change was really easy. For the second change, the display of number of threads, it took me quite some time to figure out the right routines and how to implement them correctly. Since the number of grid points are defined through the command line and printed using the 'readcmdline' routine, I thought there was also a similar way to get the number of threads.

Linear algebra kernel [15 Points]

Following the instructions I made use of OpenMP pragmas for all the possible "for"-loops in parallel. This proved to be quite efficient, so I didn't try anything further.

The diffusion stencil [10 Points]

As in the previous sub-task I used OpenMP pragmas for all "for"-loops.

2.1. Strong scaling [10+3 Points]

I plotted the time to solution for the grid sizes for 1 - 24 threads. To generate the data I used the "XeonGold 5118" Processor, 100 time steps and simulation time of 0 - 0.1s. As I generated the data for the different plots, I encountered some problems with higher numbers (> 512) of grid points. I then realized that I made a couple mistakes when implementing the stencil kernels, especially with the boundaries. After resolving those issues, I was able to get my results for 512 grid points. Unfortunately I was not able to figure out what else was wrong with my code as I did not get data for 1024 grid points. It returned an error message stating that it did not converge. Though I assume that the resulting plot would look similar to the one with 512 grid points.

For smaller grid sizes, namely 64 and 128, we can see that there is almost no difference in time regardless of the number of threads used. However if we look at bigger grid sizes as from 256 grid

points, we can see that there is a significant decrease in time after using OpenMP. The reason behind this is that there are more computations to be done for bigger problems compared to the smaller ones.

You can see my results in the following plots:

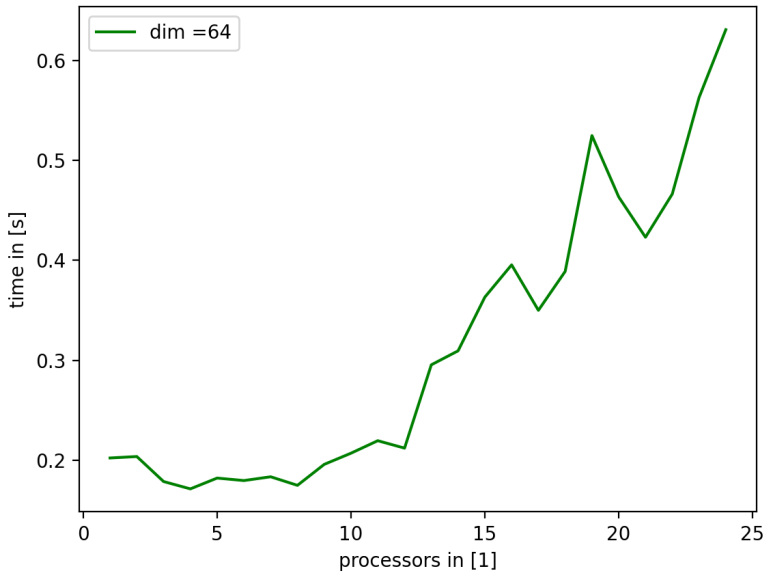

Figure 2: Strong Scaling 64x64

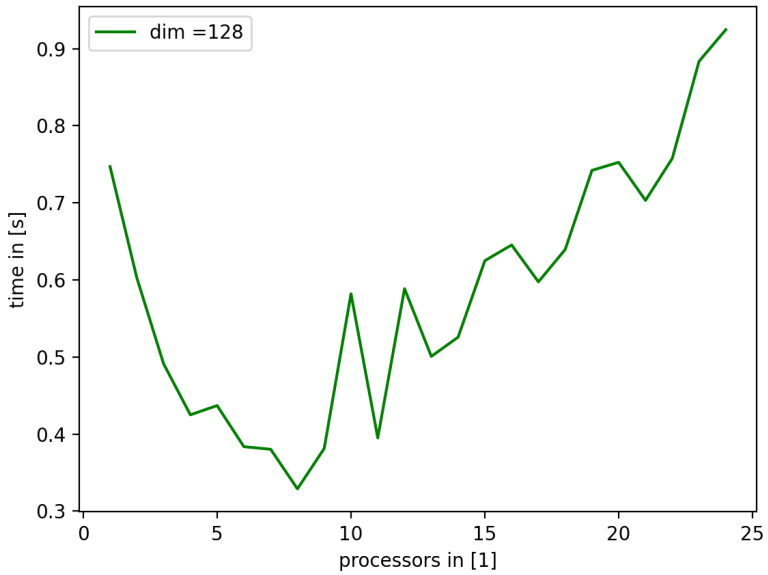

Figure 3: Strong Scaling 128x128

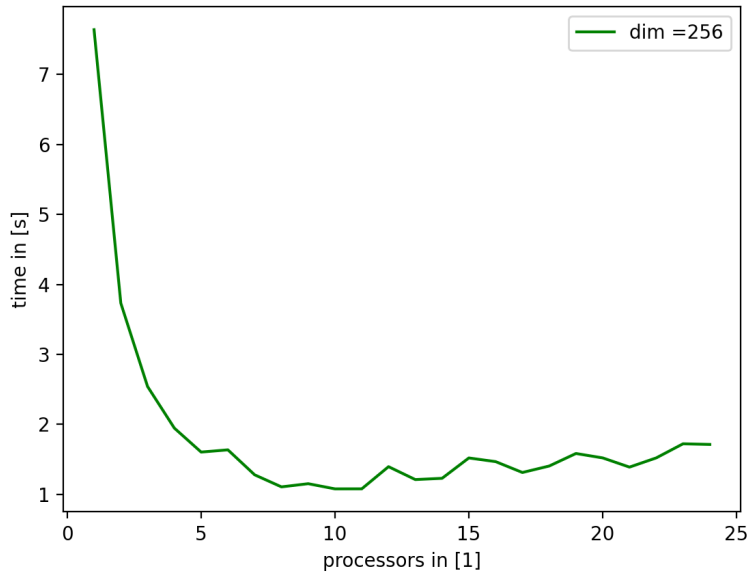

Figure 4: Strong Scaling 256x256

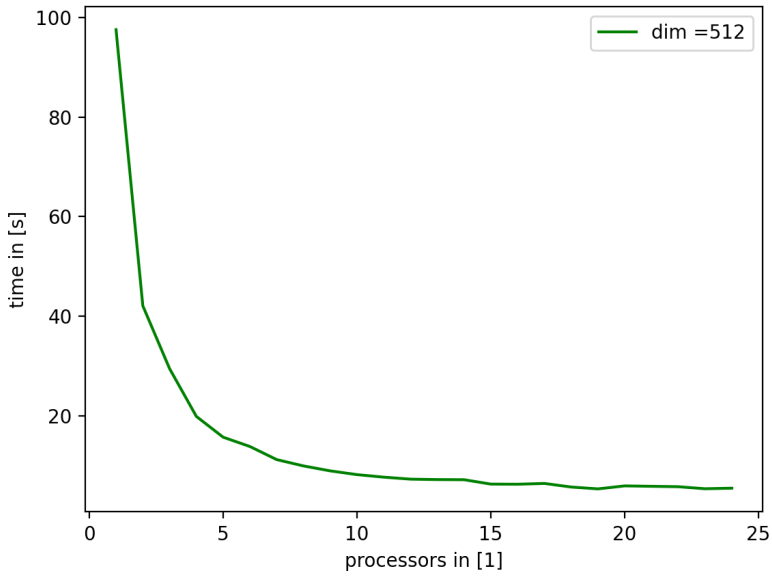

Figure 5: Strong Scaling 512x512

2.2. Weak scaling [10 Points]

To get the desired data I tried to run the simulation following the description of the task. In particular to quadruple the amount of threads when doubling the grid size. However I had troubles running the code on Euler with 64 threads. Therefore I plotted two different data sets.

First, I calculated 2048 points per thread by running 64x64 with 2 threads, 128x128 with 8 threads and 256x256 with 32 threads.

The second plot I calculated 8192 points per thread by running 128x128 with 2 threads, 256x256

with 8 threads and 512x512 with 32 threads.

In both plots we can see that it is not perfectly linear. I assume that this is on one hand due to having only very few data points and on the other hand that the scaling is rather small. However in the second plot we can already see that the results are much better than in the first plot.

You can see my results in the following plots:

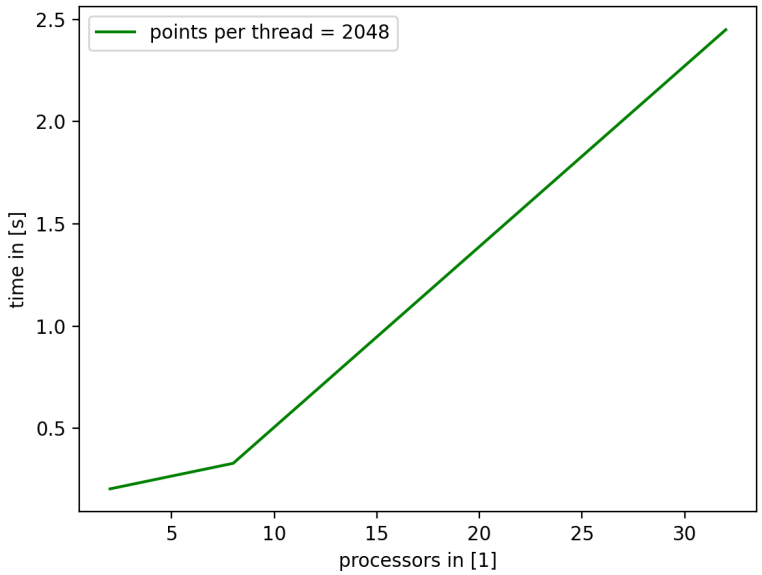

Figure 6: Weak Scaling 1

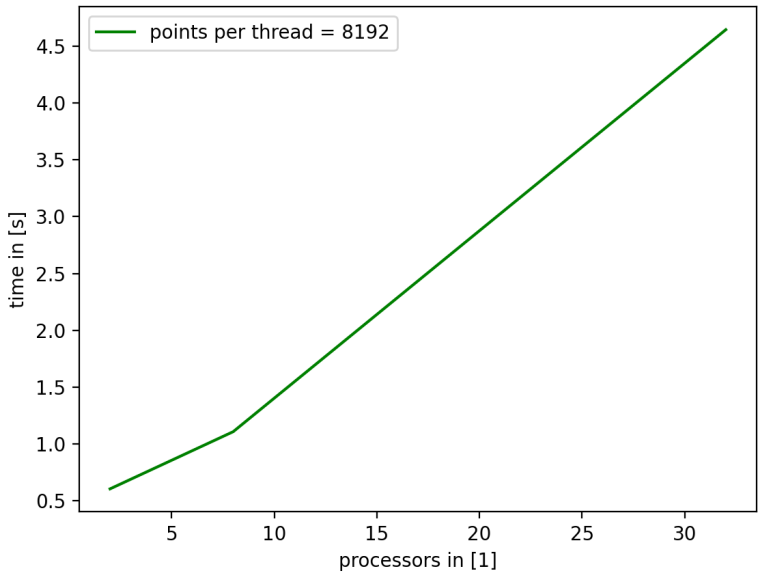

Figure 7: Weak Scaling 2

3. 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.