ETH zürich

High-Performance Computing Lab for CSE **2023**

Student: Simon Aaron Menzi Discussed with: Lukas Martin Bühler

Solution for Project 3

Due date: 10 April 2023 (midnight)

HPC Lab for CSE 2023 — 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 space solution of a nonlinear PDE using OpenMP on Euler.

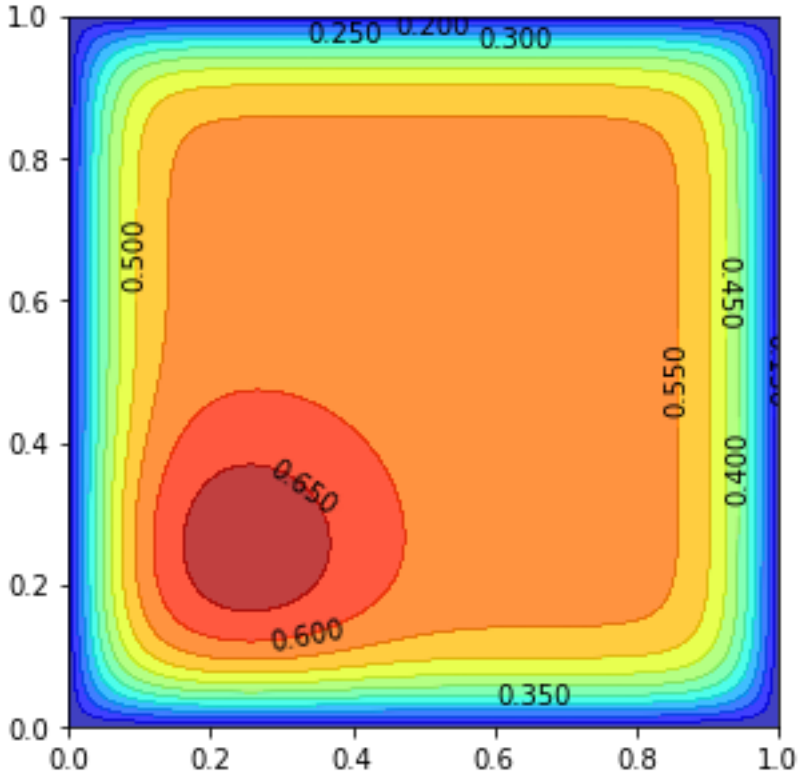
1. Task: Implementing the linear algebra functions and the stencil operators [40 Points]

1.1. Linear Algebra Implementation

The implementations were straight forward, as the task was well explained in the code as well as in the document. Simple for-loops did the job in all cases.

1.2. Stencil Implementation

As one boundary was already implemented, one only had to slightly adjust the implementation for the boundaries.



2. Task: Adding OpenMP to the nonlinear PDE mini-app [60 Points]

For all the tests, the AMD Epyc 7742 on Euler was used. The timings were done 10 times per thread number and problem size and averaged afterwards. Approaches with `simd`-pragmas were looked at and test were conducted, but always performed worse than implementations without `simd`-pragmas.

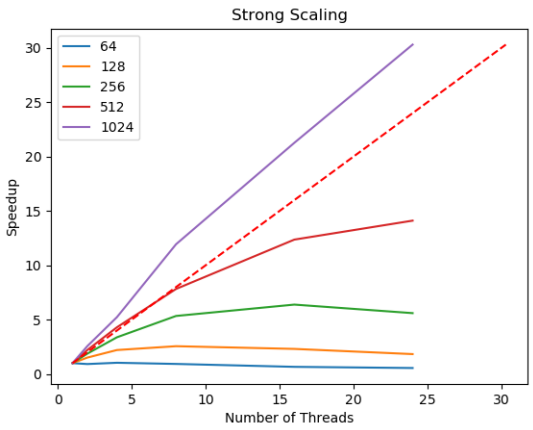
2.1. Linear Algebra Parallelization

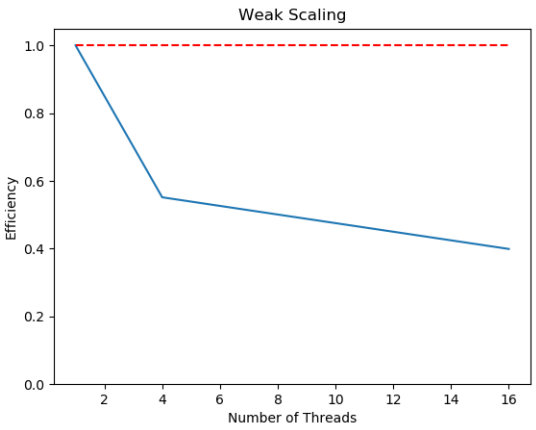
Due to the simplicity of the functions, the parallelization was an easy task. Only `#pragma omp parallel for` and `#pragma omp parallel for reduction(+:result)` were used.

2.2. Stencil Parallelization

2.2.1. Single nowait

First, a `single nowait`-approach was used. With this, the following results were achieved.

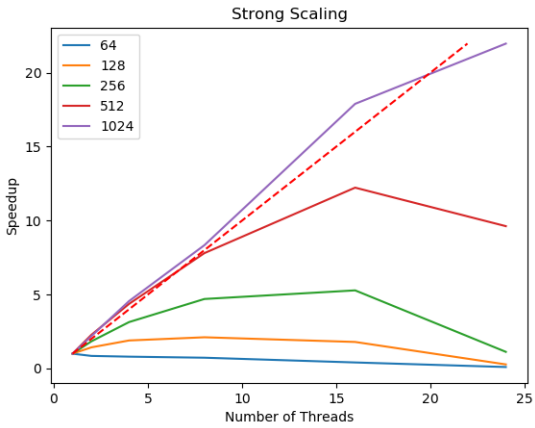


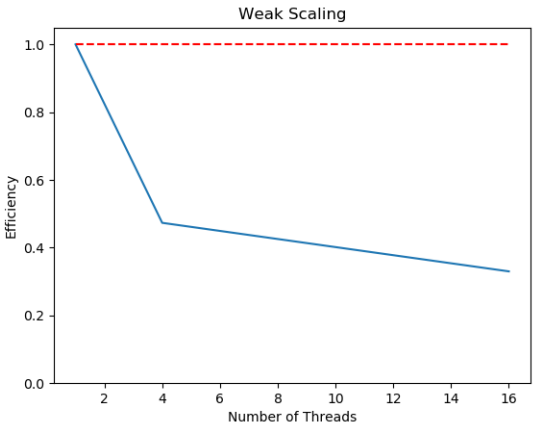


Good scaling can be observed with increasing number of threads and also with increasing problem size. How the super-linear scaling in the strong scaling plot came about is hard to explain. Weak scalability seems to be bad, as this implementations efficiency does get worse with increasing number of threads while the work load per thread stays the same. Therefore, the higher the work load on one thread, the better.

2.3. Sections

Secondly, a `sections`-approach was used. The following results were achieved.





This approach scales worse in terms of strong speedup than the `single nowait`-implementation. This can be explained with the problem itself, as more computational power can be harnessed with just going through the problem one by one and using all threads on one `for`-loop at a time, whereas a fixed number of threads have to wait after they have already finished their computations. The weak scalability test shows that this approach also does not scale well with just increasing the thread number.