ETH zürich

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

Student: Pascal Dominic Müller

Discussed with:
- Tarzis Maurer [tamaurer]
- Zuzanna Herud [zherud]

Solution for Project 6

Due date: May 20, 2021, 12pm (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. Scientific Mathematical HPC Software Frameworks - The Poisson Equation [35 points]

Note: Instead of denoting a point in Ω with (x, y) we will switch to the notation $(x_1, x_2) = (x, y)$. This makes the derivation and general notation clearer and avoids possible confusion with indices. We further use $y = g$ for the seekd after function.

1.1. Problem Statement: The Poisson Equation

We solve the poisson equation on the 2 dimensional unit rectangle $\Omega \subset \mathbb{R}^2 = [0, 1] \times [0, 1]$ with vanishing dirichlet BC.

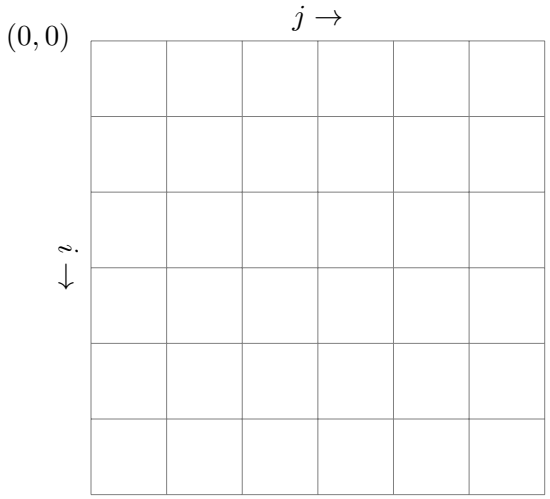
$$-\Delta g(x, y) = f(x, y) \text{ on } \Omega = [0, 1] \times [0, 1] \tag{1}$$
$$g(x, y) = 0 \qquad \text{on } \partial\Omega \tag{2}$$

whereas $\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$

1.2. Discretization

1.2.1. Grid

We discretize the domain $\Omega = [0,1] \times [0,1]$ into a set of $N+2 \times N+2$ points including the vanishing boundary. I.e. we have a grid with points $(i,j), \quad i,j \in \{0,\ldots,N+1\}$



1.2.2. Theory

Let $g(x,y)$ be an at least twice continuous differentiable function. For the discretization we use a central finite difference approach [1][Project 3].

The central finite difference in the direction of x (similar for y) is given by:

$$\delta_h[g](x) = g(x + \frac{1}{2}h) - g(x - \frac{1}{2}h) \tag{3}$$

which leads to

$$\frac{\partial g(x)^2}{\partial^2 x} = \cdots = \frac{g(x+h) - 2g(x) + g(x-h)}{h^2} \tag{4}$$

1.2.3. LHS

We first write out the laplacian operator.

$$-\Delta g(x,y) = \frac{\partial g(x,y)^2}{\partial x^2} + \frac{\partial g(x,y)^2}{\partial y^2} \tag{5}$$

Applying (4) to (5) for x and y separately, we get:

$$\frac{\partial g(x,y)^2}{\partial x^2} = \frac{g(x+h,y) - 2g(x,y) + g(x-h,y)}{h^2} \tag{6}$$

$$\frac{\partial g(x,y)^2}{\partial x^2} = \frac{g(x,y+h) - 2g(x,y) + g(x,y-h)}{h^2} \tag{7}$$

We plug (6) and (7) into (5) and use the notations (similar for y and j)

- $g(x,y) = g_{i,j}$
- $g(x+h,y) = g_{i+1,j}$
- $g(x-h,y) = g_{i-h,j}$

to end up with

$$-\Delta g^h(x,y) = \frac{g(x+h,y) - 2g(x,y) + g(x-h,y)}{h^2} \tag{8}$$

$$+ \frac{g(x,y+h) - 2g(x,y) + g(x,y-h)}{h^2} \tag{9}$$

$$= \frac{1}{h^2}(g_{i+1,j} - 2g_{i,j} + g_{i-1,j} + g_{i,j+1} - 2g_{i,j} + g_{i,j-1}) \tag{10}$$

$$= \frac{1}{h^2}(-4_{i,j} + g_{i+1,j} + g_{i-1,j} +_{i,j+1} +g_{i,j-1}) \tag{11}$$

We get:

$$\boxed{-\Delta g^h(x,y) = \frac{1}{h^2}(-4g_{i,j} + g_{i+1,j} + g_{i-1,j} + g_{i,j+1} + g_{i,j-1})} \tag{12}$$

1.2.4. RHS

On the RHS we have a function $f(x,y) : \mathbb{R}^2 \to \mathbb{R}$.

Its discretization is given by

$$\boxed{f^h_{i,j} := f(x^i, y^j)} \tag{13}$$

1.3. Discretized Problem

We end up with the following discretized problem:

$$-\Delta g^h(x,y) = f^h(x,y) \text{ on } \Omega = [0,1] \times [0,1] \tag{14}$$

$$g^h(x,y) = 0 \qquad \text{on } \partial\Omega \tag{15}$$

1.4. Implementation

For the implementation, we used PETSc. The implementation used in this report can be found in the file poisson.c. It's rather straight forward: We compute the sparse Matrix, we compute the right hand side and use a Krylov Space Solver to solve the arising linear system (14). We then output the solution and use the python script to plot it.14). We then output the solution and use the python script to plot it.

1.5. Refs

[1]: https://en.wikipedia.org/wiki/Finite_difference

Table 1: Wall-clock time (in seconds) and speed-up (in brackets) using multiple cores on Euler for solving the Poisson problem.

Problem	N	Number of Euler cores			
		1	8	16	32
Poisson	500^2	18.741 s	2.3644 s	1.6902 s	0.5808 s
Poisson	1000^2	281.5761 s	22.8054 s	25.9396 s	16.2759 s
Poisson	2000^2	1672.8654 s	227.9636 s	242.0646 s	132.6214 s
Poisson	3000^2	3913.1136 s	568.8824 s	523.6924 s	323.2356 s

2. Interactive Supercomputing using Jupyter Notebook [10 points]

Setup After going through the introduction to Jupyter Notebooks on Euler given in the problem statement for problem 3 I decided to not work with a Jupyter Notebook in the sub problem 3.2. The are two main reasons for this decision. The first one being the fact that it combines all the files into one big file, making it hard to simply work with it with the minimal default tools one has usually available, like `ed` or `vim` and similar. Furthermore I didn't like to have one big blob file in my repository.

Implementation To overcome the fact that Euler uses a python interpreter compiled with the sequential version of blas I installed numpy version 1.16.2 using `pip3 install –user numpy==1.16.2` and `threadpoolctl`.

The later is used to set the amount of maximum threads.

Implemented was a simple matrix multiplication using the @ operator. I measured the runtime for a random matrix $A \in \mathbb{R}^{10'000 \times 10'000}$.

Analysis As you can see in Figure 1, we have a very good linear speedup up to $p = 16$ cores. Surprisingly, if we increse the cores to $P = 24$ there isn't any improvement. It would not have surprised me if the speedup dropped a lot for $p = 24$ but to have no improvement surprised me. I admit this effect due to the rather small matrix size.

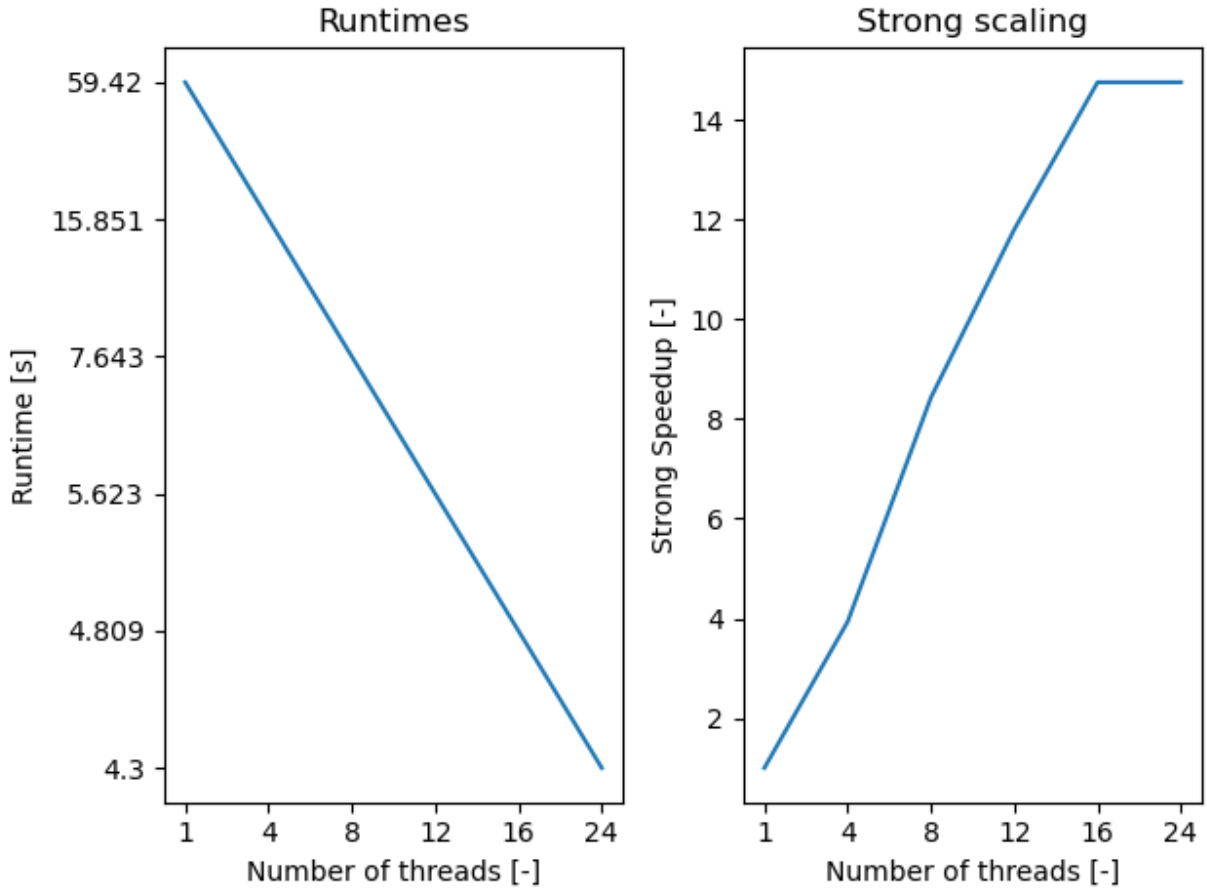


Figure 1: Strong scaling experiment for matrix multiplication.

In Figure 2 we can see the plottet solution. I didn't include plots for all the solutions but the script supports it. The difference one can see is that the surface get's smoother.

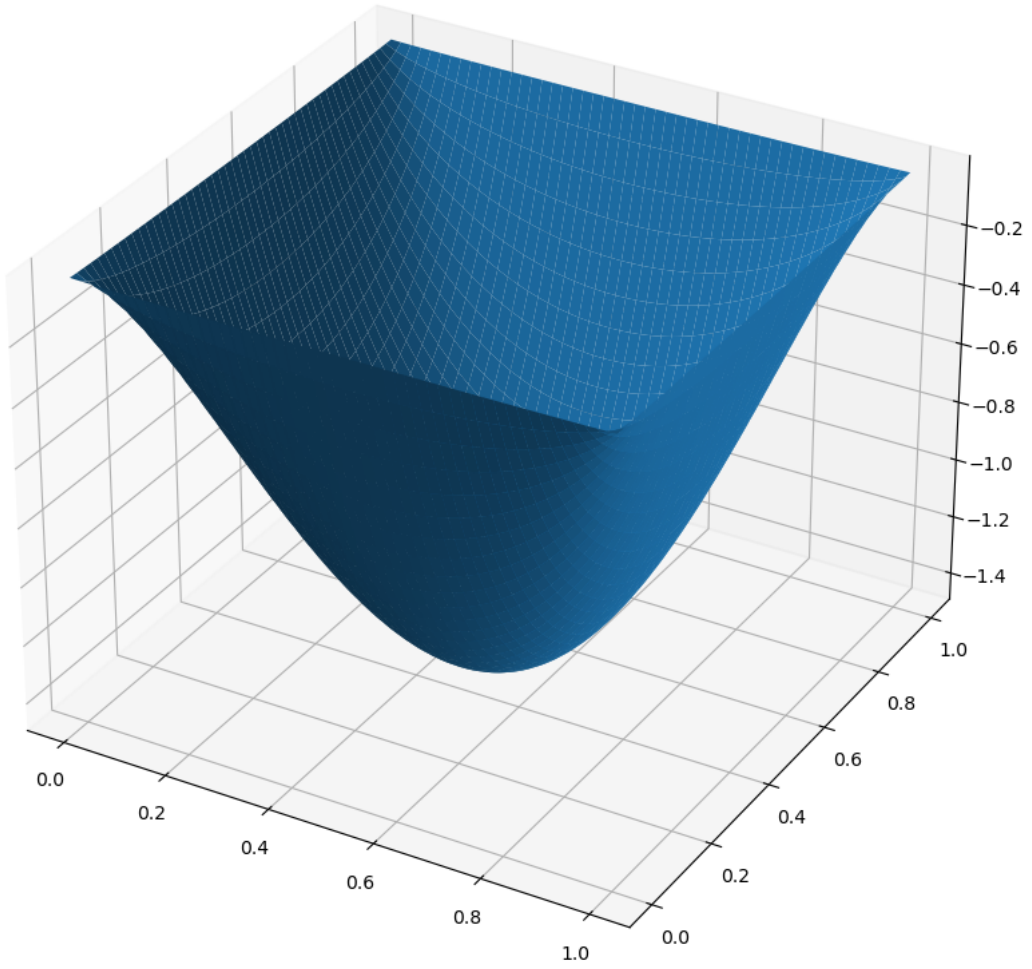


Figure 2: Strong scaling experiment for matrix multiplication.

3. Jupyter Notebook - Parallel PDE-Constrained Optimization [40 points]

We spent a lot of time on this problem but sadly we didn't have time to figure out how to properly built the hessian matrix. Thus we could not do any measurement. The code can be found in the folder "problem 4". Again, I didn't use a Notebook.

3.1. Hessian

Lagragian: $L^h(\mathbf{y}, \mathbf{u}) = \sigma F^h(\mathbf{y}, \mathbf{u}) + \lambda^T G^h(\mathbf{y})$
Hessian of Lagragian: $\nabla^2 L^h(\mathbf{y}, \mathbf{u}) = \nabla^2 \sigma F^h(\mathbf{y}, \mathbf{u}) + \lambda^T \nabla^2 G^h(\mathbf{y})$
Derivatives for $\mathbf{y}$
Note: Now $\mathbf{x} = [\mathbf{y}, \mathbf{u}]$
Diagonal Elements: $\sigma \frac{\partial^2 F^h(\mathbf{x})}{\partial x_i^2} + \lambda_{i,i} \frac{\partial^2 G^h(\mathbf{y})}{\partial x_i^2}$
Off-diagonal Elements: $\sigma \frac{\partial^2 F^h(\mathbf{x})}{\partial x_i x_j} + \lambda_{i,j} \frac{\partial^2 G^h(\mathbf{y})}{\partial x_i x_j}$

Hessian: Interior Note: $\mathbf{u}$ only affects the derivative on the boundary
$\frac{\partial F^h(\mathbf{x})}{\partial x_i} = h^2(1 - d \cdot y_{i,i}^{d-1})(y_{i,i} - y_{i,i}^d)$
$\frac{\partial^2 F^h(\mathbf{x})}{\partial x_i^2} = h^2((-d(d-1) \cdot y_{i,i}^{(d-2)})(y_{i,i} - y_{i,i}^d) + (1 - d \cdot y_{i,i}^d)(1 - d \cdot y_{i,i}^{d-1}))$

$\frac{\partial^2 F^h(\mathbf{x})}{\partial x_i x_j} = 0$

Hessian: Boundary Note: $u^d = 0 \quad \text{on} \partial \quad \Omega$
$\frac{\partial F^h(\mathbf{x})}{\partial x_i} = \alpha h(u_{i,i} - u^d_{i,i})$
$\frac{\partial^2 F^h(\mathbf{x})}{\partial x_i^2} = \alpha h^2(1 - d \cdot u^{(d-1)}_{i,i})$
$\frac{\partial^2 F^h(\mathbf{x})}{\partial x_i x_j} = 0$

Table 2: Wall-clock time (in seconds) and speed-up (in brackets) using multiple cores on Euler for solving the PDE-constrained optimization problem.

Problem	N	Number of Euler cores			
		1	8	16	32
Inverse Poisson	500^2				
Inverse Poisson	1000^2				
Inverse Poisson	2000^2				
Inverse Poisson	3000^2				

4. 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 provided on 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 solutions;
  - your write-up with your name project_number_lastname_firstname.pdf.
- Submit your .zip/.tgz through Moodle.