02616 Large-Scale Modelling - 2025 November 10, 2025

Project 2: The Poisson Problem

Your report must be handed in electronically on DTU Learn in PDF format!
Please add your source code in a ZIP file!

Deadline: Wednesday, December 3, 2025 - at midnight!

Background

Partial differential equations play an important role in many branches of science and engineering. Here we consider the Poisson problem which, in three space dimensions x, y and z, takes the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -f(x, y, z), \qquad (x, y, z) \in \Omega,$$

where $u = u(x, y, z)$ is the function we are seeking, $f = f(x, y, z)$ is a source term, and Ω is the domain in which we seek the solution. The Poisson equation describes, e.g., the steady state heat distribution in a media with constant heat capacity.
In this assignment we use the source term

$$f(x, y, z) = 3\pi^2 \sin(\pi x) \sin(\pi y) \sin(\pi z)$$

and Dirichlet boundary conditions, i.e. $u(x, y, z) = 0$ for $(x, y, z) \in \delta\Omega$, where we know the closed form solution for u:

$$u(x, y, z) = \sin(\pi x) \sin(\pi y) \sin(\pi z).$$

We can take Ω as the cube

$$\Omega = \{(x, y, z) : |x| \leq 1 \ , \ |y| \leq 1 \ , \ |z| \leq 1\}$$

and the Dirichlet boundary conditions are

$$\begin{aligned} u(x, 1, z) = u(x, -1, z) &= 0, \qquad |x| \leq 1 \ , |z| \leq 1 \\ u(1, y, z) = u(-1, y, z) &= 0, \qquad |y| \leq 1, |z| \leq 1 \\ u(x, y, -1) = u(x, y, 1) &= 0, \qquad |x| \leq 1, |y| \leq 1. \end{aligned}$$

The problem can be solved by discretization of the problem on a cubic $N \times N \times N$ grid, where we represent the solution at grid point i, j, k by the value $u_{i,j,k}$ (and similarly for f). The solution can then be computed by repeatedly updating all the *inner grid points* by means of the finite difference method and the seven-point stencil formula

$$u_{i,j,k} \leftarrow \tfrac{1}{6} \left(u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1} + \Delta^2 f_{i,j,k} \right),$$

where Δ is the grid spacing. For this problem, the solution on the boundary grid points are given by the boundary conditions, and these values are used when updating the grid points next to the boundary.

The Assignment

1. **Sequential code:** Write a (sequential) program that solves the discretized problem using the Jacobi update method. Test the program for different values of N, and familiarize yourself with the problem, the solution, and the convergence of the iterations (use the stop criterion discussed in the slides).

   Compare the iteration result with the known solution for u.

2. **Parallel Jacobi:** Implement a parallel version of the Jacobi method using `mpi4py`.

   - You can implement different domain decomposition strategies, e.g. slices along one dimension, cubes, etc. For each of the implementations, make sure that your parallel implementation produces the correct result, e.g. by comparing with your sequential version for small values of N. We expect you to use the Cartesian map for distributing domains.
   - When analysing a specific domain decomposition, motivate your choice and explain your expectations, before doing the experiments. Compare your experiment results with the expectations, and explain why — or why not — the results might deviate from your predictions. Please use scaling plots for these points.
   - Check the total performance (Mflop/s or Mlup/s, lattice updates per second) for large memory footprints, and different number of threads. Can you explain, what the limiting factor is? How does the measured performance correspond to the max. performance of the hardware used?

3. On the 18th of November, prepare a slide that details how you plan to investigate the performance/bottlenecks of your code. This will be plenum presentations. We will guide you with feedback.

Notes (for everything above):

- Use the wall-clock times when comparing different parallel test runs!
- The measured time, which will increase with larger values of N, can be converted into a performance metric, like Mlup/s (lattice updates (stencil operations) per second). This metric makes it easier to compare different experiments.
- The Mlup/s can be easily converted into memory bandwidth. How does the measured memory bandwidth compare to the hardware specifications?
- Submit your jobs to the batch system (preferred way)! Please note down the CPU type used for your experiments in the report.

- Test your parallel implementation, i.e. does it always give the same result, independent of the number of ranks.
- Fix the number of iterations to a reasonable number, when doing scaling experiments. There is no need to run until convergence is reached - your time to do experiments is limited! A good estimate for the maximum number of iterations: make it just as large, such that the sequential execution takes at most 3-4 minutes!
- Do you want a hybrid implementation? Then use multiprocessing or parallel numba and provide detailed information on how they use the computational resources.
- Use a modular structure in your program, i.e. use methods for most of the tasks. A typical structure of your main program could look like:
  1. get run time parameters from the command line, extend the template provide
  2. allocate memory for the necessary data fields
  3. initialize the fields with your start and boundary conditions
  4. call the Jacobi iterator
  5. print results, e.g. timings, data, etc.
- For visualization purposes, we provide a method that dumps a slice of the field.
- When using `Isend` or `Irecv`, they return a `Request` handle. Please ensure that you do the mandatory `Wait` or `Test` on *all* of these.

We emphasize that you shouldn't necessarily expect a progression from slower to faster performance, when going through this assignment. Rather, the goal is to try different parallelization techniques and placement of ranks for the same mathematical problem.

Consider your computer experiments as a physical experiment – details matter!