ENEL464 Embedded Software and Advanced Computing 2025
Group Assignment 2

1 Introduction

This is a group assignment with three students per group.

The purpose of this assignment is to implement a numerical algorithm on a computer to make the most efficient use of its caches and cores. You will find that you will get a large variation in program performance depending on how you implement your code and use compiler optimisations. A knowledge of computer architecture should help with this!

The algorithm is Jacobi relaxation. This is an iterative algorithm used to approximate differential equations, for example, Poisson's and Laplace's equations. Poisson's equation can be used to find the electric potential given a specified charge distribution or temperature given a specified heat source. There are more efficient ways to solve this problem using Green's functions and Fourier transforms but that is not the purpose of this assignment.

2 Jacobi relaxation

The discrete form of Poisson's equation is

$$\nabla^2 V_{i,j,k,n} = f_{i,j,k,n}, \tag{1}$$

where f_{i,j,k_n} is the source for a voxel with coordinates i, j, k at timestep n and $V_{i,j,k,n}$ is the potential field to be determined for the same voxel. If f is the source charge, V is the electric potential.

Poisson's equation is a partial differential equation and thus requires boundary conditions to achieve a solution. For the assignment, the boundaries are the six sides of a cube. The potential on the left side ($i = 0$) is -2 V, the potential on the right side ($i = N - 1$) is 1 V, and the other four sides are insulated. This is a mixed boundary condition problem since two sides have Dirichlet boundary conditions and the other sides have Neumann boundary conditions.

Poisson's equation can be solved iteratively, at each time-step n, using Jacobi relaxation, where[1]

$$V_{i,j,k,n+1} = \frac{1}{6}\left(V_{i+1,j,k,n} + V_{i-1,j,k,n} + V_{i,j+1,k,n} + V_{i,j-1,k,n} + V_{i,j,k+1,n} + V_{i,j,k-1,n} - \Delta^2 f_{i,j,k}\right). \tag{2}$$

[1] You might recognise (2) as a 3-D discrete convolution at each time-step.

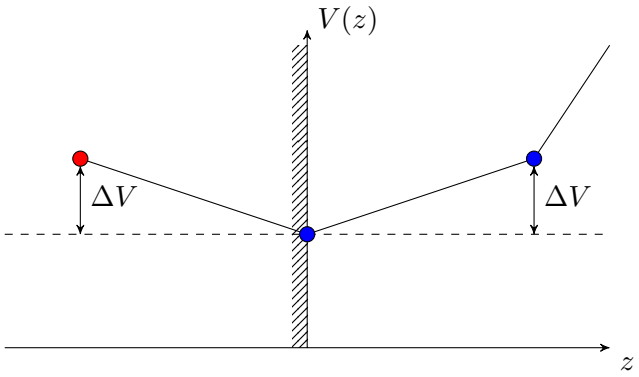

Figure 1: Illustration of a Neumann boundary condition where the external 'ghost' point (red) is defined such that the gradient at the boundary is zero with respect to the internal points (blue).

Here $\Delta = \Delta x = \Delta y = \Delta z$ is the spacing between voxels in metres, $0 \leq i < N$, $0 \leq j < N$, and $0 \leq k < N$.

Voxels on the insulated boundaries are defined such that the derivative over that boundary is zero ($\partial V/\partial n = 0$), see Fig. 1. For example, consider the derivative in the z-direction, $\partial V/\partial z$. This can be approximated by the central difference,

$$\frac{V_{i,j,k+1} - V_{i,j,k-1}}{2\Delta}. \tag{3}$$

On the boundary where $k = 0$, then

$$\frac{V_{i,j,1} - V_{i,j,-1}}{2\Delta}, \tag{4}$$

and for this to be zero implies $V_{i,j,-1} = V_{i,j,1}$. Since these voltages are the same, no current flows normal to the boundary as expected for an insulating boundary. Similarly, on the other opposite boundary, $V_{i,j,N} = V_{i,j,N-2}$.

3 Implementation

Your program to solve (2) must either use C or C++. Your goal is to find a fast implementation that will run on your (or an ECE lab) computer, making best use of the caches and multiple cores.

A good starting point is provided with `poisson.c`. You should modify this to solve the assignment. Instructions on building it are in the repository README. Some suggested reading is located in the *docs/* folder regarding compiler optimisations, multithreading, and profiling.

Code templates can be found at `https://eng-git.canterbury.ac.nz/mph/ence464-assignment-2025`. A fork of this repository will be given to you before your first lab.

We recommend writing your solutions in a Linux-like environment. This means any Linux distribution, MacOS with Homebrew, or Windows with WSL. Your mileage may vary with the profiling tools in anything other than Linux.

4 Testing

Test your algorithm with a single point charge in the centre of the volume, i.e.,

$$f_{i,j,k} = \begin{cases} 1 & i = N/2, j = N/2, k = N/2, \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

where the volume is comprised of $N \times N \times N$ voxels.

You can also test it with an arbitrary source distribution described by a set of coordinates in a text file. Instructions for the required format can be found in the repository README.

A testing script (`test.sh`) has been provided along with some sample outputs at several cube sizes and variable distributions. This will automatically compare your output against a reference implementation.

Once a correct solution has been implemented, rename `gitlab-ci.yml` to `.gitlab-ci.yml` to enable gitlab continuous integration. The pipeline will run your code against the reference solutions to confirm the correctness of your algorithm. Please do not attempt to use this for performance testing.

5 Support

Only questions submitted via the ENCE464 assignment forum will be answered. Emails will be quietly ignored.

6 FAQ

- *How do I find out the CPU version?* Run `lscpu`. You can also run `lshw` but this needs root privileges to get the cache sizes. Note, Linux considers each thread of a multithreaded core to be a CPU.
- *Why is my program killed?* This is due to the Linux out of memory killer; you have tried to allocate too much memory.

7 Reports

The reports are to be submitted as PDF documents through the ENCE464 Learn page. They will be submitted to TurnItIn for plagiarism checking. Use the supplied templates in the `docs/` folder and follow the instructions within.

Guidelines for writing a report are available at
`https://eng-git.canterbury.ac.nz/mph/report-guidelines/blob/master/report-guidelines.pdf`. However, no titlepage, abstract, introduction, or conclusion are required.

8 Assessment

The marks breakdown is:

Item	Proportion
Group report	70%
Individual report	30%
Benchmarking (bonus)	5%
Total	100%

The group report will include the following sections:

- **Architecture overview**: This should describe your computer's architecture (such as the cache organisation, memory size, CPU model, etc.).
- **Multithreading analysis**: This should discuss how your program takes advantage of multiple cores.
- **Cache analysis**: This should discuss how your program takes advantage of the cache.
- **Optimisation analysis**: This should discuss the affects of some of the compiler optimisations, such as loop unrolling, on your program.

The individual report will include the following sections:

- **Individual topic**: This should address an aspect of computer architecture related to the assignment. For example, using SIMD instructions, using the GPU, comparing two different computers, analysing branch prediction, optimisation, profiling, caching etc. in more detail. Since this is an individual assessment, the work must be completed individually, and duplicated work will be considered plagiarism.
- **Contribution statement**: A brief bullet-point list of contributions you've made to the group aspect of the project.

In addition to the quality of the content, both group and individual reports will be marked on:

- **Written style**: The writing should be concise technical writing.
- **Presentation**: The key here is consistency and clear diagrams and graphs.

Your reports should present the statistics for the time your program takes to run for the following problem sizes: $N = 101, 201, 301, 401, 501, 601, 701, 801, 901$. For $N = 801$ and $N = 901$, do not worry about performing many trials. If you have a really old computer, you can skip $N = 901$.

Your analyses should be backed up with experimental evidence, such as the output from profiling tools.

Bonus marks will be awarded to any group who can beat our (not very good) program when running on a pre-specified ESL lab computer and give the correct results. We will reveal the computer a week before the benchmarking trials. Refer to Section 9 for more details.

9 Code

We require you to use git for version control. We will look at your code if we suspect plagiarism.

Your fastest implementation must be an executable named `poisson` that is built by running `make` at the command line. For testing purposes, it must accept the following command line flags:

- `-n`: the edge length of the cube.
- `-i`: the number of iterations.
- `-s`: the path/name of the file containing coordinates for the source distribution. Your program should default to a source point with an amplitude of 1.0 at the center of the cube if this argument is not provided.
- `-t`: the number of threads to run. Your program should default to the optimal number of threads when this argument is not provided.

Your program must print out the resulting values from the middle slice of the cube in space-separated format to five decimal places. (You can just use the command line argument parsing and output formatting code in the provided `main.c`). For the benchmarking competition you must use **doubles**.

10 AI

You are permitted to use generative artificial intelligence (AI) to assist you in any way within the bounds of academic integrity. You must appropriately acknowledge any use of generative AI in your work. Please include a statement of acknowledgment with your work, clearly indicating which AI tools were used and how they contribute to your assessment.