Parallelizing the Conjugate Gradient Method with MPI

Aaron Dinesh

April 12, 2025

A project submitted in fulfilment
of the requirements for MATH-454

Declaration

I hereby declare that this work is fully my own.

Signed: Aaron Dinesh Date: April 12, 2025

1.1 Introduction

In this assignment, a serial implementation of the Conjugate Gradient Method (CGM) was provided, which required parallelization. In order to parallelize this implementation OpenMPI and the MPI parallelization paradigm was used.

1.2 Serial Analysis

The serial implementation of the CGM was evaluated using the provided `lap2D_5pt_n1000.mtx` matrix. This matrix was chosen as its large size allowed for obvious parallelization opportunities to be revealed. The `perf` profiling tool was used to analyse the serial code's execution time. From the profiling results, the `mat_vec` function was identified as the primary computational bottleneck, accounting for approximately 86.27% of the total execution time. Consequently, the serial fraction of the program was estimated to be 13.73%. The testing for the serial and parallel code were performed on the Jed Cluster with and Intel*R* Xeon*R* Platinum 8360Y CPU, running at 2.40 GHz.

These values along with Amdahl's and Gustafson's laws for strong and weak scaling were used to calculate a theoretical upper bound on the possible speed-ups that could be achieved using parallelization. The speed-ups were calculated using the following equations [1]:

$$\text{Amdahl's Law (Strong Scaling)} = \frac{1}{(1-\alpha) + \frac{\alpha}{p}}$$

Here α is the fraction of the code that can be parallelized and p is the number of processors. Gustafson's equations are as follows:

$$\text{Gustafson's Law (Weak Scaling)} = (1-\alpha) + \alpha p$$

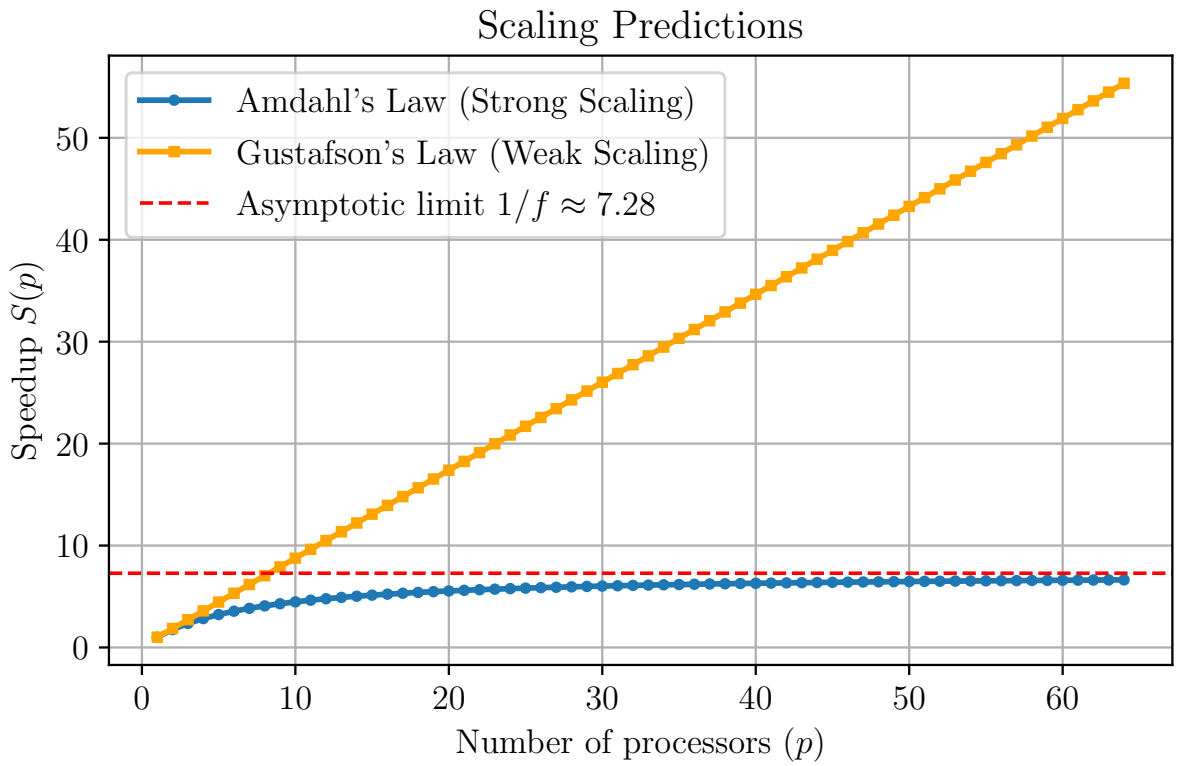

Figure 1: Strong and Weak Scaling Analysis using Amdahl's and Gustafson's Laws

1.3 Parallel Modifications

In order to parallelize the code with the least amount of communications, it was decided that the best course of action would be to read in the matrix on processor 0 and then block distribute the row index, column index and value array to each processor. Doing this when reading the matrix would implicitly parallelize the `mat_vec` function.

Bibliography

[1] D. Mishra, "A deep dive into amdahl's law and gustafson's law," 2023. [Online]. Available: https://hackernoon.com/a-deep-dive-into-amdahls-law-and-gustafsons-law