APMA2822B Assignment 2: Parallel Poisson Solver

Taj Gillin

October 18, 2024

1 Background

The goal of this assignment was to implement a similar algorithm to that done in class, but using a 2D function instead.

Before implemented a parallized version, I started with a simple version of the poisson solver in a serial fashion. Ideally, this version would set a baseline for the accuracy and performance of the parallel versions, giving me a time to compare against and an expected numerical result. This model converged in 4821 iterations, with a final L2 error of 0.278547. It took 0.484438 seconds to complete.

For all versions, I used a 2D grid of size (100×100).

For output see `out/job.78548.out`

2 Task 1: Shared Memory Model

The OpenMP model was implemented using 4 threads. The main parallelization efforts focused on:

1. Initializing the grid and setting boundary conditions 2. Updating interior points in the main computation loop 3. Calculating the L2 distance for convergence check 4. Updating the solution for the next iteration

This parallel implementation yielded identical results to the serial version, converging in 4821 iterations with a final L2 error of 0.278547. The overall execution time was reduced to 0.0786312 seconds, with an average time of 1.7×10^{-5} seconds per 1000 iterations, or 1.7×10^{-8} seconds per iteration.

The time spent in parallel regions was 0.040118 seconds, which is 51.0205

- Main computation loop: 55.3977 GB/s - Convergence error calculation: 36.2605 GB/s

These results are visualized in the roofline model below.

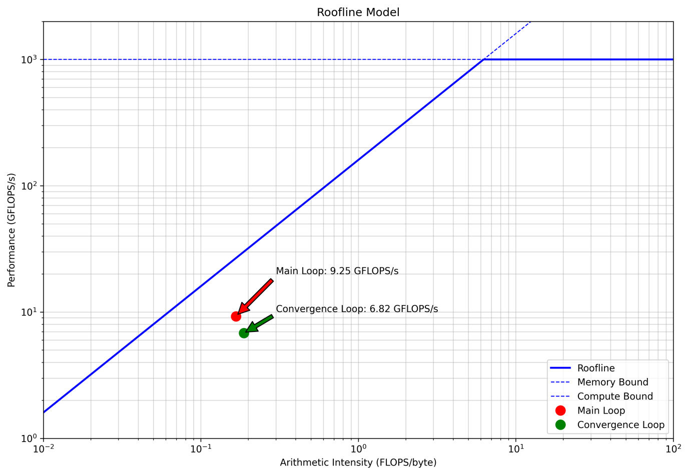

Figure 1: Roofline model for the OpenMP version

For output see `out/job.78747.out`

3 Task 2: MPI Model

The MPI implementation was more complex and quite tricky, requiring a domain decomposition approach. The key aspects of this implementation include:

1. Domain Decomposition: The 2D grid was divided into smaller subgrids, with each process responsible for a portion of the domain. This was achieved using a 2D Cartesian communicator.

2. Ghost Cell Exchange: To handle the boundaries between subdomains, a ghost cell exchange mechanism was implemented. Each process communicates with its neighbors to update the ghost cells before each iteration. This was done using `MPI_Isend` and `MPI_Irecv`.

3. Local Computations: Each process performs computations on its local subdomain, including updating interior points and calculating local L2 distances.

4. Global Communications: MPI collective operations (`MPI_Allreduce` and `MPI_Reduce`) were used to combine local results for global convergence checks and final error calculations.

The implementation steps were as follows:

1. Initialize MPI and create a 2D Cartesian communicator
2. Determine local domain size for each process
3. Allocate memory for local arrays, including ghost cells
4. Initialize local data and set boundary conditions
5. Implement the main iteration loop:
   (a) Exchange ghost cells with neighboring processes
   (b) Update interior points of the local domain
   (c) Calculate local L2 distance
   (d) Perform global reduction to check convergence
   (e) Update solution for the next iteration
6. Calculate final L2 error using local computations and global reduction
7. Output results (from rank 0 process)

This approach allows for efficient parallelization across multiple nodes, with each process working on a portion of the domain and communicating only necessary boundary data. The ghost cell exchange ensures that each process has the required information from its neighbors to perform accurate computations on its subdomain.

Unfortunately, I was not able to get the *exact* same results, but the results are very very close! The program converged in 4963 iterations, with a final L2 error of 0.275526. It took 0.0429326 seconds to complete, making it $\sim 45\%$ faster than the OpenMP version.

For output see `out/job.78738.out`