CS 426/525: Project 2

Spring 2025

Introduction

In this assignment, you will implement the conjugate gradient (CG) method in C for solving linear systems and parallelize it using the Message Passing Interface (MPI). The focus will be on dense matrices, to understand the challenges and performance implications of parallelizing standard linear algebra operations over distributed systems.

Background

The conjugate gradient method is an algorithm for the numerical solution of particular systems of linear equations ($Ax = b$), whose coefficient matrix is symmetric. The method is often used in computer simulations for solving the systems of linear equations arising in the discretization of partial differential equations.

For a detailed understanding of the conjugate gradient method, please refer to the first and second documents listed in the additional resources section. These documents provide foundational knowledge and advanced insights necessary. For specific guidance on the serial and parallel implementations of the conjugate gradient method, it is recommended to read sections 2 and 3 of the paper listed in item 3 of the additional resources. These sections delve into the practical aspects of coding the method.

Input Generation

The conjugate gradient method solves systems of linear equations where the coefficient matrix is **symmetric** and **positive definite**. An easy way to generate a positive definite matrix is to generate a diagonally dominant matrix. A square matrix is said to be diagonally dominant if, for every row of the matrix, the magnitude of the diagonal entry in a row is greater than or equal to the sum of the magnitudes of all the other (off-diagonal) entries in that row. More precisely, the matrix A is diagonally dominant if

$$|a_{ii}| \leq \sum_{j \neq i} |a_{ij}|$$

where a_{ij} denotes the entry in the ith row and jth column.
So, the generated matrices must be

- **Square**
- **Symmetric**
- **Diagonally dominant**

You can generate the p vector with positive, double-precision random numbers. (It would be better if you use a range between 0-1)

Part A (Serial)

In this section, you will implement a serial conjugate gradient solver. While doing this, I recommend reading the second section (specifically, Algorithm 1) of the paper listed as item 3 in the additional resources.

Directives and Notes

- Read the inputs, start the timer, perform the computation, end the timer, and write the output vector to a file.
- Name of the source code file and the output file (txt or out) must be serial.c and s_output.txt/out, respectively.

Part B (Parallel)

In this part, you will parallelize the conjugate gradient solver you implemented serially, using MPI for a distributed memory setting. For this part, I recommend carefully reading Section 3 of the paper listed as item 3 in the additional resources and implementing your solution by following Algorithm 3.

The part you will parallelize is the matrix-vector multiplication in the conjugate gradient solver ($q = Ap$). The rows of the dense matrix A and the vector p will be distributed across the processors. As shown in the figure, P_2 owns the 3rd and 4th rows of the matrix and the corresponding elements in the vector, and it is responsible for computing the 3rd and 4th entries of the output vector q. For example, to compute q_3, P_2 needs to multiply row 3 of the matrix with the entire vector p. For the entries of p that it does not own, P_2 must communicate with the other processes to obtain them. In this example P_2 must receive $p_{1,2}$ from P_1, $p_{5,6}$ from P_3, and $p_{7,8}$ from P_4. (You can refer to the paper for more details.)

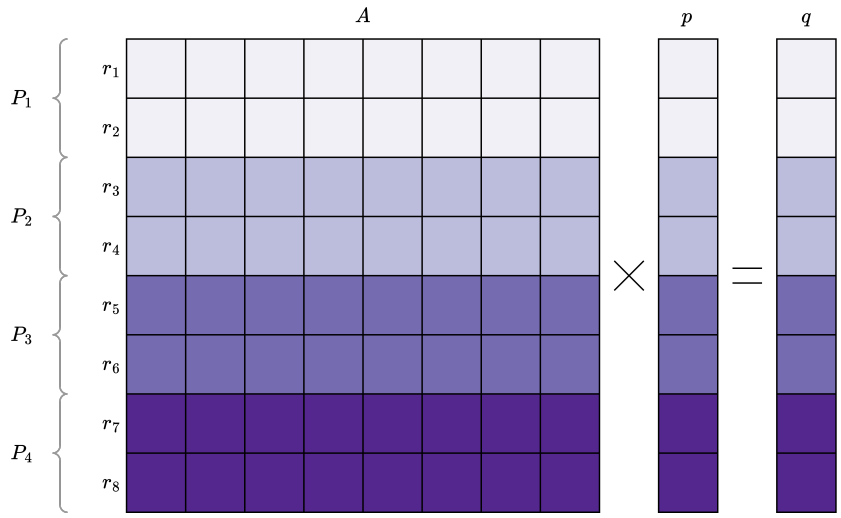

Figure 1: Row-parallel matrix-vector multiplication

Directives and Notes

- After reading the input matrix and vector, the master process should distribute them to the other processes. It should also collect the final output vector from the other processes and write it to a file. **In this project, the master process should not participate in computation, it should only coordinate input/output operations.** (You may assume that the number of rows in the matrix and vectors is divisible by the number of processors.)
- Communication between processes/cores during the matrix-vector multiplication operation must be P2P.
- You should start the timer after the initial distribution phase is finished and end it just before the final reduction of the output vector.
- Name of the source code file and the output file (txt or out) must be parallel.c and p_output.txt/out, respectively.

Experiments

- Run your serial and parallel implementations on ORFOZ.
- Run your parallel implementation on 16, 32, 64, 128, and 256 processes/cores and observe the runtimes while process/core count increases.

- Provide at least 2 strong scaling curves with different input sizes. (From 1 core (serial) to 256 cores)
- Provide at least 2 weak scaling curves. (Discussed in detail in the class)
- Use MPI_Wtime function for time measurements and do not forget to put necessary barriers while measuring the time.

Submission Format

Submit a single zip file containing source files, a Makefile, a SLURM script file, sample input/output files (the smallest ones), and a project report discussing your findings to Moodle. name_surname_p2.zip file should include the followings:

- Your implementation with source files and necessary inputs for the following:
  - serial.c
  - parallel.c
- A makefile that generates the following 2 executables:
  - serial
  - parallel
- Your report (4 pages at most):
  - Briefly explain your implementations. Plot graph(s) with various thread numbers for small, medium, and large input sizes indicating the performance of your implementation by comparing serial and parallel. What are your observations? Does parallel implementation improve performance? If not, what might be the reasons?

Grading

- Serial Implementation: 25 points
- Parallel Implementation: 50 points
- Report: 25 points

Important Notes

- **DEADLINE: April 27, 2025 23:59**
- **No Late Submission Allowed!**

- Your submission file **MUST BE** in a **ZIP** format. (not RAR, TARBALL etc.)
- In order for your project to be graded, it must be able to compile with the Makefile you provided and must run without any problem on the TRUBA/ARF system with the SLURM script file you provided.
- You can always reach out to me via email. (can.bagirgan@bilkent.edu.tr)

Additional Resources

- An Introduction to the Conjugate Gradient Method Without the Agonizing Pain
- Wikipedia
- R. O. Selvitopi, M. M. Ozdal and C. Aykanat, "A Novel Method for Scaling Iterative Solvers: Avoiding Latency Overhead of Parallel Sparse-Matrix Vector Multiplies," in IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 3, pp. 632-645, March 2015.