Introduction to Parallel Computing.
Homework 2: Parallelizing matrix operations using MPI.

a.y. 2024/2025

Abstract

This assignment is intended to evaluate your effort, knowledge, and originality. Although you may discuss some issues related to the proposed problem with your classmates, each student should work **independently** on their assignment and provide a short report. The methodology used, the quality of the deliverable and its originality (with respect to the state of the art and to the classmate's solutions) will be assessed.
Below, we describe the requirements for the report:

1. Author: Student Name, Surname, ID, email
2. Maximum 4 pages, excluding the references section
3. References
4. Platform and computing system description
5. **GIT link** and instructions for reproducibility. The results must be perfectly reproducible, so the repository must contain a **README file** with instructions to reproduce the results (instructions for the compilation and execution) and the code.
6. Format: use IEEE conference template (2-column format, main text 10pt)
7. The report should summarize the solution to the problem and explain the methodology used in clear details.
8. To submit the report, please complete the following form and ensure all necessary information is included by **January 19, 2025**, at 23.59.

Disclaimer: If the report does not meet one or more requirements, it will NOT be evaluated.

1 Problem Statement: Parallel Matrix Transposition

In this assignment, you will explore explicit parallelization techniques using Message Passing Interface (MPI) by implementing a matrix transpose operation. You will benchmark and analyze the performance of this approach, comparing its efficiency and scalability with the parallel approaches of the first homework. Consider a matrix M of size $n \times n$, where n is a power of two.

Task 1: Sequential Implementation

1. Write a C/C++ program to perform sequential matrix transposition. The program should:
   - Initialize a random $n \times n$ matrix M of floating-point numbers.
   - Implement a function `checkSym` to determine if the matrix is symmetric.
   - Implement a function `matTranspose` to compute the transpose of M and store the result in matrix T.
   - Use wall-clock time to measure the execution times of both `checkSym` and `matTranspose`.
   - Take the matrix size n as an input parameter.

You may reuse the sequential implementation from Homework 1, if it was correct.

Task 2: Parallelization with MPI

1. Implement a parallel version of matrix transposition, `matTransposeMPI`, using MPI. Experiment with MPI routines and explore different algorithms to improve the performance.
2. Implement also a parallel version of the symmetry check, `checkSymMPI`.

Task 3: Performance Analysis

1. Evaluate the performance of your implementations: sequential and MPI parallelism
2. Measure the time taken for matrix transposition for varying matrix sizes n from 2^4 to 2^{12}
3. Compute and discuss speedup and efficiency gains for the MPI implementation, with different matrix sizes and number of processors. Consider both weak and strong scaling. Consider the performance metrics that is more suitable for this exercise.
4. Identify potential bottlenecks or problems and propose optimizations to help solve them.

Task 4: Comparison of Parallelism Approaches

1. Evaluate and compare the performance of all implementations (sequential, OpenMP, MPI) for varying matrix sizes n from 2^4 to 2^{12}. You can compare the performance, scalability, and ease of implementation. Highlight the key advantages and limitations of each approach.
2. Analyze the speedup and efficiency of each parallel approach relative to the sequential baseline. Discuss:
   - OpenMP performance for varying numbers of threads.
   - MPI performance for varying numbers of processes.
   - Scalability of both approaches.
3. Identify and discuss bottlenecks or inefficiencies in each approach. Provide recommendations for improving scalability and efficiency.

Bonus Task (Optional): Parallelization by blocks with MPI

1. Implement a new function `matTransposeBlock` in your sequential code to compute the transpose of M in blocks as illustrated in the provided figure

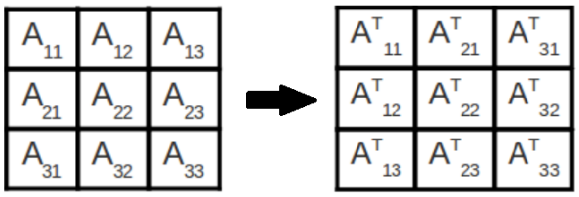

Figure 1: Block-based matrix transposition.

2. Implement a parallel version of matrix transposition by blocks, `matTransposeBlockMPI`, using MPI. Use a 2D block decomposition to:
   - Distribute blocks of the matrix among processes.
   - Each process computes its assigned block(s) of the transposed matrix.
   - Use MPI communication routines to exchange matrix data.
3. Experiment with different block sizes and different MPI routines to identify the optimal configuration for performance on your hardware.

4. Evaluate the performance of your block-based implementation and compare with the one without blocks

Report organization

Follow these guidelines for structuring your report:

- **Abstract:** A brief summary of the project, highlighting objectives, methods, and findings.
- **Introduction:** Background, importance of the problem, and objectives of the project.
- **State-of-the-art:** Review current research and developments related to your project, discuss existing solutions and their limitations, and identify the gap your project aims to fill.
- **Methodology:** Detailed descriptions of your implementations, including algorithms (pseudo-code), data structures and parallelization strategies. Discuss the challenges faced and how they were addressed.
- **Experiments and System Description**
  - Detailed description of the computing system and platform.
  - Relevant specifications or configurations (e.g., libraries and programming toolchains).
  - Description of the experimental setup, procedures, and methodologies used in the project.
  - Discussion on how experiments are designed to test the hypotheses or achieve the objectives.
- **Results and Discussion:** Present, analyze and interpret the results, including performance metrics, speedup, and efficiency. Compare with the state-of-the-art.
- **Conclusions:** Summary of findings and contributions.
- **GIT and Reproducibility:** Include the GIT repository link and detailed instructions in a README file.
- **References:** List of references used.