Introduction to Parallel Computing.
Homework 3: Parallelizing matrix operations using MPI.

STUDENT NAME:
EMAIL:
STUDENTID:
Git link:

a.y. 2023/2024

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.

Guidelines:

1. The report should summarize the solution to the problem and explain the methodology used in clear details.
2. Experiments must be carried out in the **UniTn HPC cluster**.
3. Please submit the document together with a link to the git repository to us by **December 13, 2023**, at 23.59 at the following emails: `flavio.vella@unitn.it`, `laura.delrio@unitn.it` and `anas.jnini@unitn.it`, with e-mail subject **IntroPARCO 2023 H3**.
4. Experiments must be perfectly reproducible, so the repository must contain a **README file** with the instructions to reproduce the results (instructions for the compilation and execution), the code and the report.

Please choose between exercise 1 and 2 and perform only one of them. Exercise 3 is mandatory

1 Parallel matrix multiplication

The first exercise consists of implementing a parallel matrix multiplication program using MPI. The tasks to be performed are:

1. Serial matrix multiplication:
   To implement a serial matrix multiplication algorithm, write a C/C++ sequential program that contains a function `matMul` that takes two input matrices, A and B (of random float numbers), and returns their product, C. Be sure to validate the compatibility of the matrices for multiplication. Use a wall-clock time, at the program level, that measures the time of the matrix multiplication only.

2. Parallel matrix multiplication:
   Parallelize the matrix multiplication using MPI. Create a function `matMulPar` that takes the same inputs as the serial version and parallelizes the algorithm. Experiment with different MPI functions and data partitions to improve performance. Explore different parallelization strategies to find the most efficient implementation.

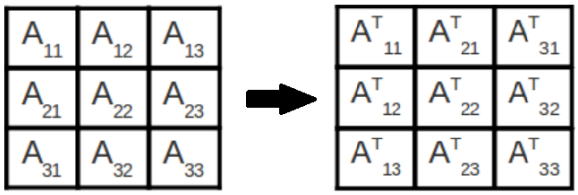
3. Parallel Block-based matrix multiplication:
   Implement a parallel matrix multiplication by dividing the matrix into blocks and distributing these blocks across MPI processes. Create a function `matMulBlockPar` that takes the same inputs as the serial version and parallelizes the algorithm using MPI. Experiment with different block sizes and MPI functions to optimize performance.

4. Performance analysis:
   Assess the performance and scalability of your parallel implementations with MPI. Compute speedup and efficiency gains for different matrix sizes and number of processors (consider both weak and strong scaling). **Consider for this task the time of the routine and compute the FLOPS as performance metrics**. Identify potential bottlenecks or problems and propose optimizations to help solve them. Compare the performance of the normal multiplication and the block-based multiplication.

2 Parallel Matrix Transposition

This exercise consists of writing a program for parallel matrix transposition using MPI. The program must efficiently handle matrices of real numbers. The tasks to be performed are:

1. Sequential code:
   Write a C/C++ sequential program with the matrix transpose algorithm for matrices of real numbers. Define a function `matT` that takes an input matrix A and returns its transpose. Define a second function `matBlockT` that takes an input matrix A and returns its transpose, calculating the transpose in blocks as shown in the picture.



2. Parallel matrix transpose:
   Parallelize the matrix transpose operation using MPI for matrices of real numbers. Create a `matTpar` function that takes the same input as the serial version and parallelize the algorithm. Experiment with MPI functions to improve performance. Explore different algorithms to find the most efficient one.

3. Parallel transpose of block matrices:
   Implement a parallel matrix transpose by dividing the matrix into blocks and transposing these blocks. Create a function `matBlockTpar` that takes the same input as the serial version, divides the matrix into blocks, and computes its transpose using the block division. Each block must be transposed, and the final transposed matrix is constructed from these blocks. Experiment with different block sizes and different MPI functions to optimize performance.

4. Performance analysis:
   Evaluate the performance and scalability of both parallel matrix transpose methods (with blocks and without blocks). Calculate the speedup and efficiency gains for different matrix sizes and number of processor (consider both weak and strong scaling). **Consider for this task the time of the routine and compute the bandwidth as performance metrics**. Identify possible bottlenecks or problems and propose optimizations. Compare the performance of normal transpose and block-based transpose.

3 Comparison of Parallelism Approaches

Finally, you should compare the three parallelism approaches you have learned in the course: implicit parallelism, OpenMP, and MPI. Use, for example, the matrix multiplication problem to compare the approaches: you can compare the performance, scalability, and ease of implementation. Explore how these approaches perform on different hardware architectures.
You must provide clear information about the results and justify your answer. The report should contain information about the performance analysis, including speedup, efficiency, and any proposed

optimizations. It is important to explore and compare different implementations to identify the best approach. Include the **bibliography** that you have used at the end of your report.