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

STUDENT NAME:
EMAIL:
STUDENTID:

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. The simulations are to be launched on the **UniTn HPC cluster**.
3. Please submit the document together with a link to the git repository to us by **November 22, 2023**, at 23.59 at the following emails: `flavio.vella@unitn.it`, `laura.delrio@unin.it` and `anas.jnini@unitn.it`, with e-mail subject **IntroPARCO 2023 H2**.
4. The simulations performed 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) and the code.

1 Parallel matrix multiplication

The first exercise consists of implementing a parallel matrix multiplication program using OpenMP. The program must be able to handle efficiently both dense and sparse matrices of real numbers. The tasks to be performed are:

1. Serial matrix multiplication:
   Implement a serial matrix multiplication algorithm. You will need to define a `matMul` function that takes two input matrices, A and B, and returns their product, C. Be sure to validate the compatibility of the matrices for multiplication.

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

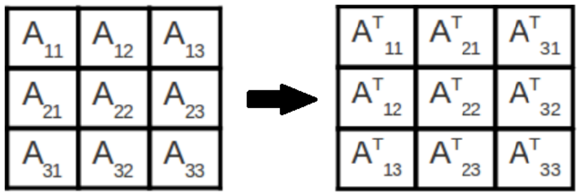
3. Performance analysis:
   Evaluate the performance and scalability of your parallel implementation of matrix multiplication. Calculate speedup and efficiency gains for different matrix sizes and number of threads. **Consider for this task the time of the routine and compute the FLOPS as performance metrics**. Identify potential bottlenecks or problems and propose optimizations to help solve them.

Describe the differences between the parallel algorithms for dense and sparse matrices, if any.

2 Parallel matrix transposition

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

1. Serial matrix transposition:
   Implement a serial 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.


A_{11} A_{12} A_{13}
A_{21} A_{22} A_{23}
A_{31} A_{32} A_{33}
A^T_{11} A^T_{21} A^T_{31}
A^T_{12} A^T_{22} A^T_{32}
A^T_{13} A^T_{23} A^T_{33}

2. Parallel matrix transpose:
   Parallelize the matrix transpose operation using OpenMP for matrices of real numbers. Create a `matTpar` function that takes the same input as the serial version and parallelizes the algorithm. Experiment with OpenMP directives 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 OpenMP directives 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 thread counts. **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. Discuss the differences that may appear in parallelization algorithms when you are working with dense or sparse matrices.

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.