Introduction to Parallel Computing.
Homework 1: Exploring Implicit and Explicit Parallelism with OpenMP.

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 what you need to do concerning the project 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 the 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. Please submit the document together with the git repository link to us by **December 1, 2024**, at 23.59 at the following emails: flavio.vella@unitn.it and laura.delrio@unitn.it, with e-mail subject **IntroPARCO 2024 H1**.

Disclaimer, if the report does not match one or more requirements it will NOT be evaluated.

1 Problem Statement: Parallel Matrix Transposition

In this assignment, you will explore both implicit and explicit parallelization techniques by implementing a matrix transpose operation. You will benchmark and analyze the performance of both approaches, comparing their efficiency and scalability. 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, that performs a sequential matrix transposition. The program should:
   - Initialize a random $n \times n$ matrix M of floating-point numbers.
   - Implement a function `checkSym` that compute if the matrix is symmetric
   - Implement a function `matTranspose` that computes the transpose of M and stores the result in a new matrix T.
   - Use wall-clock time to measure only the execution times of the symmetry check (`checkSym`) and matrix transposition routine (the `for loop`).
   - Take the matrix size n as an input parameter.

Task 2: Implicit Parallelization

1. Implement a second function, `matTransposeImp`, that improves the performance of your matrix transpose using implicit parallelization techniques such as vectorization, prefetching, or optimizing memory access patterns.
2. Do the same for parallelizing `checkSym`.

Task 3: Explicit Parallelization with OpenMP

1. Write a function `matTransposeOMP` that parallelizes the matrix transpose operation using OpenMP.
2. Use OpenMP directives to parallelize the computation. Experiment with work-sharing or block-based transposition strategies to improve performance and find the most efficient implementation.
3. Use OpenMP directives for accelerating `checkSym`, call this routine `checkSymOMP`

Task 4: Performance Analysis

1. Evaluate the performance of your three implementations: sequential, implicit parallelism, and OpenMP 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 OpenMP implementation, with different matrix sizes and number of threads, compared to the sequential baseline. Consider the performance metrics that is more suitable for this exercise. Identify potential bottlenecks or problems and propose optimizations to help solve them.
4. If applicable, discuss the effect of compiler flags and implicit parallelization techniques on performance.

Bonus Task (Optional): Peak Performance Comparison

- Compare the performance of your matrix transpose routines against the peak performance of the system. Identify the theoretical peak memory bandwidth of the system and include a plot comparing your routines against this peak.

You must provide clear information about the results and justify your answers. The report should contain information about the solution proposed, 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.

1.1 Report organization

Recommendations on how to organize the report.

Abstract

- Summary of the project, highlighting key objectives, methods, and findings.

Introduction of the problem and importance

- Describe the background and importance of the problem.
- Outline the main objectives of your project.

State-of-the-art

- Review current research and developments related to your project.
- Discuss existing solutions and their limitations.
- Identify the gap your project aims to fill.

Contribution and Methodology

- Elaborate on the unique contributions of your project.
- Describe the methodology proposed, including algorithms (pseudo-code), data structures, and parallelization techniques.
- 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

- Presentation of results.
- Analysis and interpretation in context.
- Comparison with the state-of-the-art.

Conclusions

- Summary of key findings and contributions.

GIT and Instructions for Reproducibility

- GIT repository link the point to the code and the detailed instructions for reproducing results (README).