Homework 2: Programming with OpenMP

Suhaib Abdi Muhummed, muhummed@kth.se
Abdurahman Ahmed, abdahme@kth.se

Course code: ID1217 VT2025
Date: 2025-02-08

1 Compute the Sum, Min, and Max of Matrix Elements

The purpose of this problem is to introduce the to basic OpenMP usage. The provided program ('matrixSum-openmp.c') computes the sum of matrix elements in parallel using OpenMP. The task is to modify the program to:

- Compute the sum of all matrix elements.
- Find and print the minimum and maximum matrix element values along with their positions (indexes).
- Implement the modifications using OpenMP constructs.
- Evaluate performance across different numbers of processors (at least 4).
- Report the speedup (sequential time divided by parallel execution time).

1.1 Performance Evaluation

- **Workers = 2 vs. 4:** The computation time reduces slightly when increasing from 2 to 4 workers, but the difference is inconsistent for small matrices due to parallelization overhead.
- **Workers = 4 vs. 8:** The performance improves noticeably for larger matrices.
- **Workers = 8 vs. 16:** The improvement continues, but diminishing returns appear due to synchronization overhead and memory bandwidth constraints.

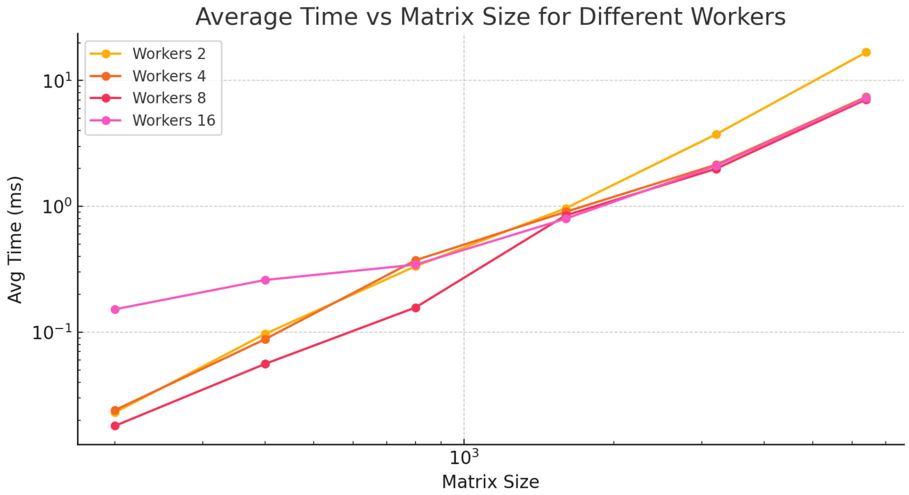

Figure 1: Performance of Compute Sum, Min, and Max using OpenMP

2 Quicksort

This task involves implementing parallel Quicksort to sort a list of numbers. The list is divided into two sublists where:

- All numbers in the first sublist are smaller than those in the second sublist.
- The sorting is performed in parallel using OpenMP.
- The program runs on different processor configurations (at least 4).
- Speedup is calculated as sequential time / parallel execution time.
- Tests are conducted on at least 3 different list sizes.

2.1 Performance Evaluation

- **Workers = 2 vs. 4:** Speedup values range from 1.0 to 1.7, showing benefits but not strong scaling due to overhead.
- **Workers = 8:** Speedup improves significantly, reaching 2.2 to 2.4 across different list sizes.
- **Workers = 16:** Speedup increases to 2.4 to 2.7, but additional workers offer diminishing returns due to synchronization overhead and memory bandwidth limitations.

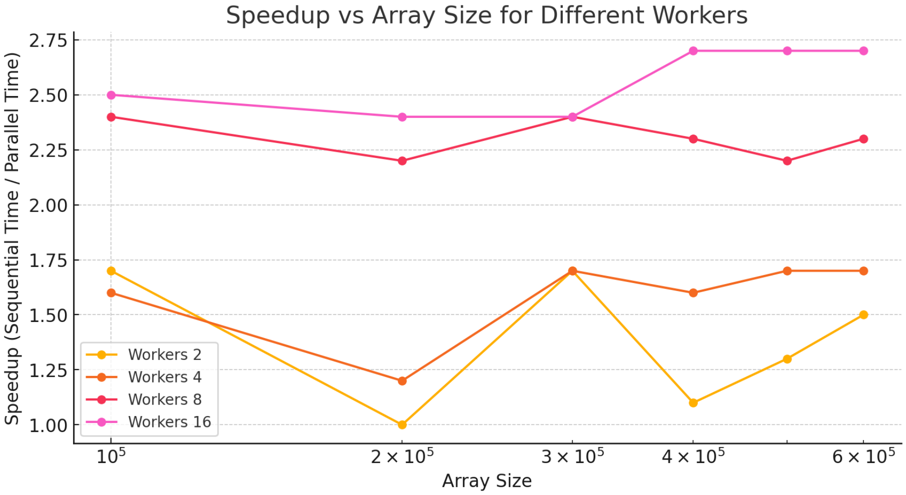

Figure 2: Performance of Parallel Quicksort using OpenMP