Programming assignment 3

Getting started

This is a continuation of the previous assignments. You have already worked with these algorithms in sequential and shared-memory/GPU settings; in this assignment, you will implement them using distributed-memory parallelism with MPI.

The problems below should be solved using C/C++ and MPI. You are expected to use a recent MPI implementation (e.g. Open MPI or MPICH) and compile with `mpicc`/`mpicxx`.

You are allowed to use any external sources you can find. The code you submit must be written by members of your group, but it may be inspired by online material. All group members must be able to explain all parts of the submitted solution.

Tasks

As in Assignment 1, you will work with two algorithms:

1. Odd–Even Sort
2. Pearson Correlation Coefficient (PCC)

Use the same sequential reference implementations from Assignment 1 (`oddevensort`, `pcc_seq`). Do not modify the reference implementations; instead, add MPI versions (e.g. `oddevensort_par_mpi.cpp`, `pcc_par_mpi.cpp`) and use the sequential programs for validation and performance comparison.

Task 1 – MPI Odd–Even Sort

Implement an MPI version of odd-even sort. The algorithm works by alternating phases where independent pairs of elements can be compared and swapped in parallel. In an MPI setting, this requires coordinated communication and synchronization between ranks (for example, exchanging boundary elements or blocks with neighbor ranks between phases). A sorted output is guaranteed after n iterations of the main loop. Note that the input may already be fully sorted in an ith iteration of the loop, where i < n. The algorithm could be written to check during every iteration whether the input is fully sorted, but we do not implement that optimization in this task.

Requirements:

- Validate correctness for arbitrary input sizes and process counts (a simple loop that verifies $v[i] <= v[i+1]$ should be enough)
- Discuss communication/synchronization overhead and how it impacts performance

Task 2 – MPI Pearson Correlation Coefficient

Your task is to implement an MPI version of the Pearson correlation coefficient, corresponding to the same PCC task from Assignment 1.

PCC generates the input matrix on the fly from matrix dimensions and an optional seed (no separate input file); both sequential and parallel PCC write correlation output to files that can be compared with the provided verify program.

The Pearson correlation coefficient (Wikipedia) measures the linear relationship between two datasets. It ranges between -1 and $+1$, where 0 implies no correlation. A correlation of -1 or $+1$ implies an exact linear relationship. A positive correlation means that as x increases, y increases. A negative correlation means that as x increases, y decreases.

The correlation coefficient is calculated as follows:

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}}$$

where m_x is the mean of vector x and m_y is the mean of vector y (SciPy documentation).

The PCC computation involves multiple reductions that can be parallelized across MPI ranks.

Requirements:

- Implement an MPI PCC computation
- Ensure correctness for arbitrary input sizes and process counts within a reasonable floating-point tolerance (use the provided verify program to check for correctness; the benchmark script runs it automatically)

Performance requirements and grading

This assignment is about correctness, parallelization, and performance reasoning.

Benchmarking. For odd–even sort, benchmark scalability using an array of 2^{19} elements; you may also use other sizes. For PCC, benchmark with matrices up to 4096×4096.

To receive a pass grade, your submission must demonstrate:

- Correct results for both algorithms
- Positive speedup (> 2.0) for PCC on at least one non-trivial input size

Higher grades will be awarded based on:

- Quality of parallel implementation
- Clear and reproducible benchmarking methodology
- Insightful performance analysis
- Discussion of communication, synchronization, load balance, and scalability limits

There is no fixed speedup requirement for higher grades, but results should be reasonable given the algorithmic characteristics.

Submission guidelines

The submission consists of two parts: a demo and a code/report submission.

Demo

When your group is ready, you can join the online lab sessions and demonstrate the assignment. During the demo, you should:

- Run your code on a machine you bring
- Demonstrate correctness for both algorithms
- Explain your parallelization approach and performance results

You may benchmark on another machine, but your code must be runnable during the demo.

Code and report

After passing the demo stage, submit your solution via Moodle.

You may work in groups of one to three students. All submitted material must be written by the group. Only one group member should submit to Moodle, but all names must be listed in the report.

Your submission must include:

- Source code (including your MPI implementations; the provided Makefile has targets for `oddevensort_par` and `pcc_par`—add your MPI source files and adjust the Makefile if needed)
- A Makefile to compile your programs. You may use the one provided as part of the sequential version.
- A report describing:
  - implementation choices (parallelization strategy, data/task decomposition, synchronization mechanisms, ...)
  - benchmarking setup (hardware and software setup, number of MPI ranks, input sizes, # of runs, ...)
  - performance results (speedup, execution time, ...)
  - explanation of observed (scaling) behavior

Deadline

Submit your solutions as a single ZIP file via Moodle no later than end of day March 8, 2026.

You must complete the demo before submitting.