

# Programming assignment 1

## Getting started

The problems below should be solved using C++. You are expected to use a recent compiler version, such as GNU GCC (v13 or above) or LLVM/Clang (v18 or above).

If you run Linux, install the compiler via your package manager. If you use Windows, we strongly recommend using Windows Subsystem for Linux (WSL), which works well together with VSCode. On macOS, you can use the Clang compiler provided by the developer tools. Note that Apple's Clang does not support OpenMP by default. You may still complete the assignment using threads, but if you want to use OpenMP you must install it separately.

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

In this assignment, you will work with two algorithms:

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

For both algorithms, correct sequential reference implementations are provided (`oddevensort`, `pcc_seq`). You must not modify the reference implementations; instead, add your parallel versions (e.g. `oddevensort_par.cpp`, `pcc_par.c` or `pcc_par.cpp`) and use the sequential programs for validation and performance comparison.

### Task 0 – Familiarization and baseline validation

Your task is to:

- Compile and run the provided sequential implementations (`make` builds `oddevensort` and `pcc_seq`)
- Understand the control flow and data access patterns of both algorithms

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.

This task serves as preparation for the parallel implementations. No new code needs to be written for this task, but you are expected to fully understand the baseline implementations before continuing.

### **Task 1 – Multithreaded Odd–Even Sort**

Implement a multithreaded version of odd–even sort. This algorithm works by alternating phases where independent pairs of elements can be compared and swapped in parallel. Correct synchronization between phases is required. A sorted output is guaranteed after  $n$  iterations of the main loop. Note that the input may already be fully sorted in an  $i$ th 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:

- Implement a parallel odd–even sort using threads, OpenMP, or a combination of both
- Validate correctness for arbitrary input sizes and thread counts (A simple loop that verifies  $v[i] \leq v[i + 1]$  should be enough)

### **Task 2 – Multithreaded Pearson Correlation Coefficient**

Your task is to implement a multithreaded version of the Pearson correlation coefficient.

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 threads.

Requirements:

- Implement a parallel PCC computation
- Ensure correctness for arbitrary input sizes and thread 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  $4096 \times 4096$ .

To receive a pass grade, your submission must demonstrate:

- Correct results for both algorithms
- Positive speedup ( $> 2.0$ ) for both odd–even sort and 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 synchronization, memory behavior, 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 parallel implementations; the provided Makefile has targets for `oddevensort_par` and `pcc_par`—add your 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, thread counts, 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.