Any part that says (Milestone 1) means that I need to come back and write in general for what has been done

# Assignment 1 – Report

Name: Tyler Roe-Smith

Student ID: a1899603

Repository: https://github.com/Mealsise/P-and-D-computing/tree/assignment-1

## Introduction

This report documents the implementation and design of an MPI-based program for identifying out-of-order values in a ring of communicating processes. Each process holds an integer and compares it with the value of its immediate left neighbour to determine if it is in order.

The assignment is divided into two parts:

* numbers1.c: Each process independently checks its own value and prints a message if it is out of order.
* numbers2.c: Out-of-order values are reported to a master process (rank 0), which prints ordered output and a summary count.

The program uses point-to-point communication (MPI\_Send and MPI\_Recv) to simulate a ring topology, where each process can only communicate with its immediate neighbours.

## Input Handling

At this stage of the assignment, each process allocates and accesses a hardcoded list of values. Although the final implementation may involve reading input externally or distributing data across processes, the focus for Milestone 1 is on establishing correct communication and logic.

All processes initialize the same fixed array of integers. Each process then selects the value at the index corresponding to its rank and compares it with its left neighbour’s value using MPI point-to-point communication. This approach ensures consistent behaviour for testing while verifying that neighbour-based messaging and value comparisons function as intended.

## Approach & Design

The program was designed around a ring communication model, where each process can only send to and receive from its immediate neighbours using point-to-point MPI communication. The goal was to identify whether each process holds a value that is out of order relative to its left neighbour.

To achieve this:

* Each process initializes with a hardcoded list of integers.
* Based on its rank, it selects the corresponding value from the list.
* All processes (except rank 0) receive the value of their left neighbour using MPI\_Recv.
* Each process (except the last) sends its value to the right neighbour using MPI\_Send.
* A comparison determines if the current value is less than the left neighbour’s value, indicating it is out of order.

This design falls under **SIMD (Single Instruction, Multiple Data)** in Flynn’s classification. All processes execute the same logic in parallel but operate on different input data (i.e., their assigned numbers). Each process follows the same communication pattern and decision-making structure independently.

## Ording in First Program

(Milestone 1) Ordering is not maintained and thus the outputs order is nondeterministic across processes Each process prints whether its value is in order (“all good” or “no good”) followed by a termination message (“I’m done”). While the sequence within each process is correct, there is no guarantee of global ordering between processes.

## Interaction Diagram – First Program

(Milestone 1) Ring diagram with point-to-point communication. Each process communicates only with immediate left and right neighbours. The edge from process 0 to the last rank (wrap-around) is unused in this version.

## Ordering in second Program

(Milestone 1) Local comparisons will be (not yet copied across from q1) the same as in the first program. However, only **rank 0 prints output** after receiving messages from other ranks. Each process relays a message through the ring to rank 0, which prints results in rank order.

## Interaction Diagram – Second Program

(Milestone 1)

Rank 0 acts as the master

Processes relay messages LEFT to rank 0

Messages include packed metadata:

* 1 bit: pass/fail
* 3 bits: sender rank
* Remaining bits: failing value

Optionally, an array of ints can be used if input exceeds 2²⁸

Rank 0 unpacks and prints results in order