Computação em Larga Escala

Practical Assignment 2

João Monteiro (102690), João Sousa (103415), João Gaspar (107708)

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

Contents

1 Introduction

This project aims to analyze and compare the performance of two implementations, single-threaded and distributed-memory MPI—applied to two specific problems: *Weather Stations* and *Word Count*. By migrating sequential solutions to MPI-based distributed applications, we evaluate scalability across multiple processes in a cluster environment.

1.1 Problem 1: Weather Stations

The *Weather Stations* problem involves reading meteorological data from various stations and computing statistics such as average, minimum, and maximum temperature per station. The sequential implementation processes all samples in one process, while the MPI version partitions the input among multiple MPI ranks, each computing local statistics and then gathering and sorting results at the root rank.

1.2 Problem 2: Word Count

The *Word Count* problem counts words, lines, and characters in a collection of text files. The sequential version processes each file end-to-end, whereas the MPI version distributes file paths across ranks; each rank performs counts locally and the root process aggregates the totals.

1.3 Objective

The primary objective is to implement distributed-memory versions of both applications using MPI, measure execution times across 1, 4, 8, and 16 processes, and analyze speedup and efficiency to assess scalability and overheads.

2 Implementation Logic

In both MPI implementations, the core strategy follows a three-phase pattern: data partitioning, local computation, and global aggregation/synchronization.

2.1 Data Partitioning

Each MPI process (rank) is assigned a contiguous chunk of the input to process independently:

- **Weather Stations:** We use MPI I/O to determine the total file size, split it into equal byte ranges per rank, and seek each process to its start offset. To avoid partial records, non-zero ranks skip the first line if it falls within an unfinished record.
- **Word Count:** The root rank reads the list of filenames, broadcasts the total count and each filename to all ranks, then divides the file indices among ranks (handling uneven remainders) so each rank processes an exclusive subset.

2.2 Local Computation

Once partitioned, each rank performs its share of the work independently:

- **Weather Stations:** Each line is parsed into `<station;temperature>`, and a local `unordered_map<string,WSData>` accumulates temperature statistics (min, max, total, count) per station.
- **Word Count:** Each rank invokes a local `word_count` function on its assigned files, tallying characters, lines, and words into a map of `CountResult` per filename.

2.3 Global Aggregation and Synchronization

To consolidate distributed results, both applications use MPI collectives:

- **Weather Stations:**
    1. Pack each station's statistics into a custom `MPI_Datatype` representing `StationData` (char name[128] + WSData fields).

2. Perform an `MPI_Gather` of selected pivot entries for sorting and bucket boundaries.
3. Use `MPI_Bcast` to share pivot values for range partitioning.
4. Redistribute all local entries via `MPI_Alltoallv` into buckets determined by comparison to pivots.
5. Each rank merges its bucket entries by summing WSData for identical station names and then sorts its final segment locally.
6. Rank 0 prints the fully ordered list.

- **Word Count:**

  1. Serialize each file's `CountResult` into a text buffer ("file;chars;lines;words").
  2. Use `MPI_Gather` to collect the sizes of each buffer, then `MPI_Gatherv` to assemble all serialized data at rank 0.
  3. Rank 0 deserializes the aggregated buffer into a global map, prints per-file and total counts, and reports overall execution time using `MPI_Wtime`.

3 Performance Analysis

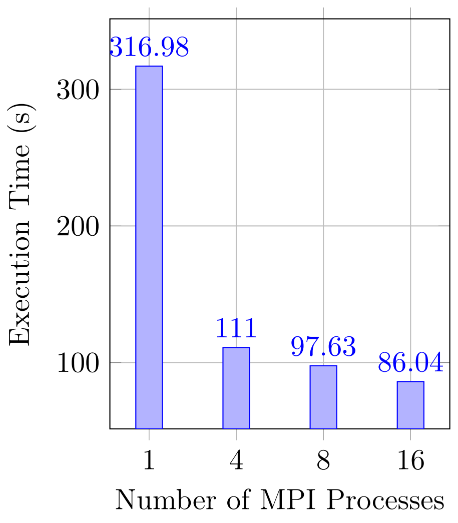
To evaluate distributed performance, we executed both applications on an input containing *one billion* weather samples and a text file list for word count. Baseline measurements use one MPI process (`mpirun -np 1`), and subsequent runs use 4, 8, and 16 processes.

3.1 Measured Execution Times

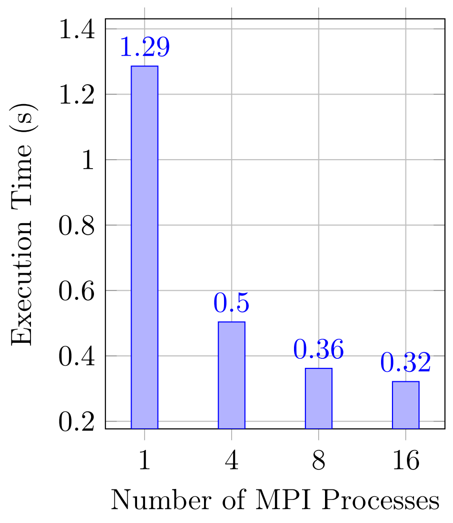
The recorded execution times (in seconds) are:

- **Weather Stations (1 000 000 000 samples):**
  - 1 process: 316.978 s
  - 4 processes: 110.997 s
  - 8 processes: 97.6261 s
  - 16 processes: 86.038 s

- **Word Count (file list `files.txt`):**
  - 1 process: 1.28606 s
  - 4 processes: 0.503809 s
  - 8 processes: 0.361848 s
  - 16 processes: 0.321504 s



(a) Weather Stations



(b) Word Count

Figure 1: Execution Time vs. Number of MPI Processes for both applications

3.2 Speedup Calculations

Speedup is defined as $S_p = T_1/T_p$, where T_1 is the time with one process and T_p with p processes.

- **Weather Stations:**
  - 4 processes: $316.978/110.997 \approx 2.86$
  - 8 processes: $316.978/97.6261 \approx 3.25$
  - 16 processes: $316.978/86.038 \approx 3.68$
- **Word Count:**
  - 4 processes: $1.28606/0.503809 \approx 2.55$
  - 8 processes: $1.28606/0.361848 \approx 3.56$
  - 16 processes: $1.28606/0.321504 \approx 4.00$

3.3 Test Conditions and Assumptions

- **Input Data:**
  - Weather Stations: input file with **1 billion** samples in "Station;Name;Temperature" format.
  - Word Count: list of text filenames in `files.txt`, covering approximately 641 books encoded in UTF-8.

- **MPI Configurations:** Experiments used 1, 4, 8, and 16 MPI processes on a single node.
- **Hardware:** 13th Gen Intel Core i9-13980HX 2.20GHz, 32GB RAM (24 cores/32 threads).

4 Discussion

The distributed-memory MPI implementations demonstrate clear performance gains over the one-process baseline. For Weather Stations, speedup increases from 2.86× (4 processes) to 3.68× (16 processes), indicating moderate scaling, with diminishing returns likely due to communication and synchronization overhead. In contrast, Word Count exhibits steeper scaling, reaching 4.00× at 16 processes, reflecting more coarse-grained workload distribution and lower inter-process communication.

5 Conclusion

Implementing MPI-based distributed solutions significantly accelerates both applications. The Word Count problem benefits more from added processes, while the Weather Stations application encounters greater overhead.