

Parallel and Distributed Computing

**Project Report**

**Submitted to: Sir Fahad Shafique**

**Submitted By: Muhammad Ali Naveed  
Hamza Omer**

**Ariyan Chaudhary**

[**Parallel Implementation of the Bubble Sort Network using MPI and OpenMP**](#_h170v8cwmc5d) **2**

[1. Introduction](#_1p3p2wrfbht9) 2

[2. Algorithm Description](#_7ob8g6b7niv) 2

[3. MPI + OpenMP Parallel Implementation](#_goa7us77cl8i) 2

[3.1 MPI Process Division](#_ahu8xer8ero7) 2

[3.2 OpenMP Thread Division](#_4s4n1brm4mnc) 3

[3.3 Timing and Output](#_994ci93q5pyt) 3

[4. Correctness and Validation](#_1pn4ijd5k0th) 3

[5. Experimental Setup](#_q1slbacmeq17) 3

[5.1 Cluster Configuration](#_3cw1fvbeop7m) 4

[5.2 Homelab Server](#_y05frjxbw99c) 4

[6. Performance Analysis](#_it6rfxz36jc) 4

[6.1 Tools Used](#_ahurkehkvo17) 4

[6.2 Scalability Observations](#_kap8bn8zxsjt) 4

[6.3 Chart: Execution Time vs Processes](#_a06zioaj8i1r) 5

[6.4 Chart: Speedup Comparison](#_qj7r2o3bc90v) 5

[7. Observations and Challenges](#_u7iknlbnmdya) 5

[8. Conclusion](#_sjssz1xa52he) 5

[A. Compilation and Execution Instructions](#_2ynf7fk0hedo) 6

[B. Sample Output File Format](#_8ialynl1886) 6

# 

# **Parallel Implementation of the Bubble Sort Network using MPI and OpenMP**

## **1. Introduction**

This report presents the design, implementation, and analysis of a parallel algorithm based on the *bubble sort network* for generating parent permutations of the symmetric group Sn. The algorithm follows the model outlined in [Chitturi et al.], where permutations are organized using an adjacency network akin to sorting networks, and each permutation’s parent is determined via swap operations governed by network rules.

We parallelized the algorithm using **MPI** for process-level distribution across nodes and **OpenMP** for multi-threaded computation within each MPI process. This hybrid model improves scalability and computation time for larger values of n, where the total number of permutations grows factorially.

## **2. Algorithm Description**

The core algorithm determines, for each permutation in Sn, the parent permutations according to a layered, bubble sort-inspired swap network.

* For each permutation (vertex), n−1, parent permutations are computed by conditional swaps.
* The parent-finding logic implements complex rules as per the referenced paper, using position inversion arrays and ranking/unranking of permutations.
* The result for each permutation is stored as a CSV row containing the vertex ID, the permutation string, and the parent IDs for each transformation layer.

## **3. MPI + OpenMP Parallel Implementation**

### **3.1 MPI Process Division**

The factorial space Sn of permutations is evenly divided among the available MPI processes:

* Each process receives a distinct chunk of permutations to analyze.
* Load balancing is achieved using the division and remainder trick (base = Nperm / size; rem = Nperm % size;), ensuring fair distribution.
* Each process writes its results to a separate CSV file stored in /tmp/results/.

### **3.2 OpenMP Thread Division**

Each MPI process internally uses OpenMP to further divide its chunk across threads:

* Threads independently compute parent permutations.
* Buffering is done locally per thread using a vector<vector<string>> thread\_lines.
* A single write operation after the barrier ensures consistent file output with minimized I/O contention.

### **3.3 Timing and Output**

* MPI\_Wtime() is used to record per-process runtime.

Output files contain CSV data in the format:  
vertex\_id,perm,T1,T2,...,T(n-1) for all assigned permutations.

## **4. Correctness and Validation**

* The implementation closely follows the specification and pseudocode of the original paper .
* We ensured correctness by comparing ranks and unranking outputs, verifying that the parent permutations correspond to expected transitions.
* The is\_identity() function and custom swap\_symbol() logic ensure accurate parent tracing.

## 

## **5. Experimental Setup**

### **5.1 Cluster Configuration**

* **Total Nodes:** 3
* **Node Specs:**
  + Node A: 8 GB RAM
  + Node B: 16 GB RAM
  + Node C: 32 GB RAM
* **OS:** Ubuntu Server 22.04 LTS
* **MPI Implementation:** OpenMPI 4.1.5
* **Compiler:** GCC 11.3.0 with mpicxx
* **Networking:** Ethernet (some setup overhead noted)

### **5.2 Homelab Server**

* **Single Node Specs:** 64 GB RAM, 12-core CPU
* **Results:** Better performance and lower latency due to unified memory access and no inter-node overhead.

## **6. Performance Analysis**

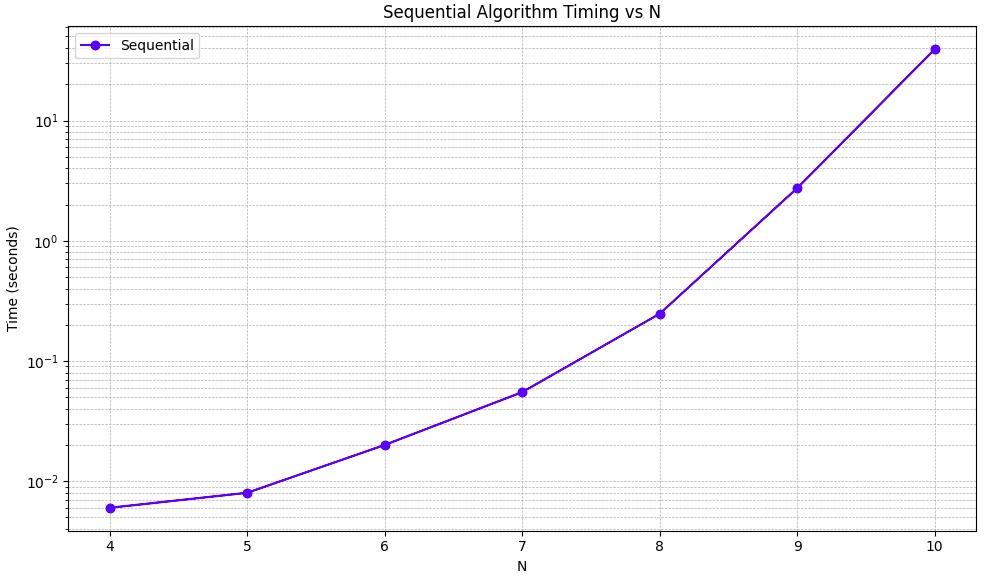
### **6.1 Tools Used**

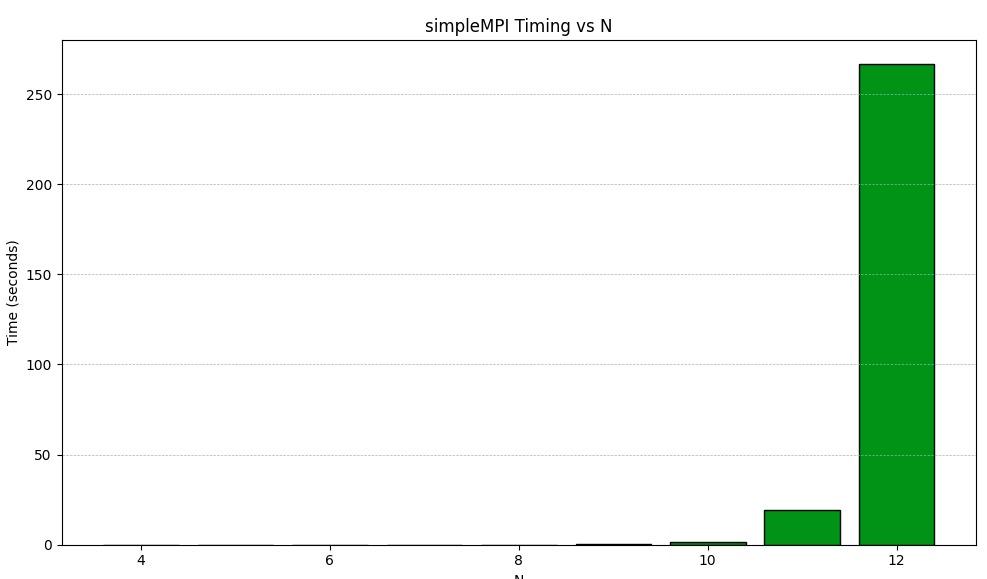
* gprof for profiling sequential versions.
* MPI\_Wtime() and std::chrono for measuring time per process/thread.
* Plots and speedup charts will be inserted here.

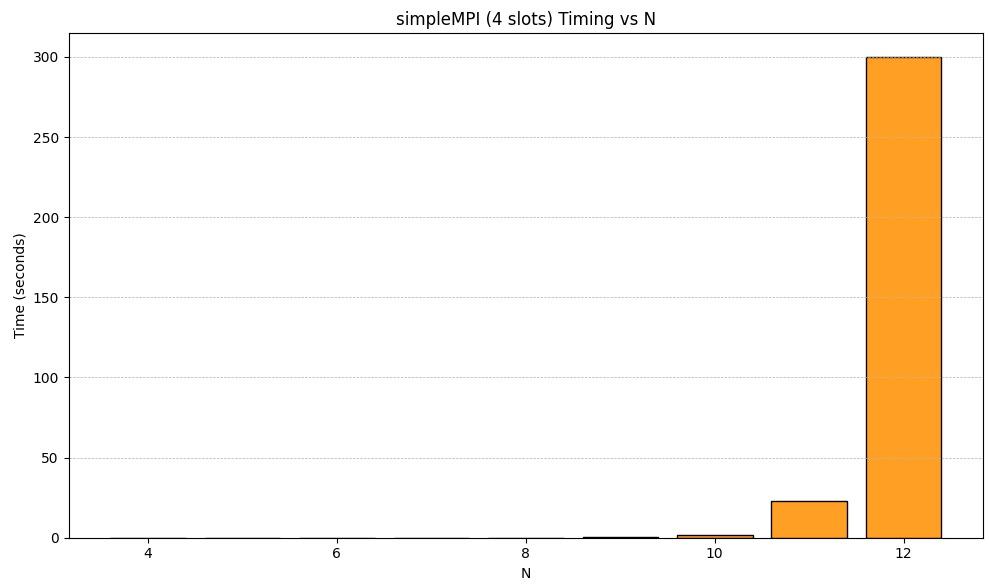
### **6.2 Scalability Observations**

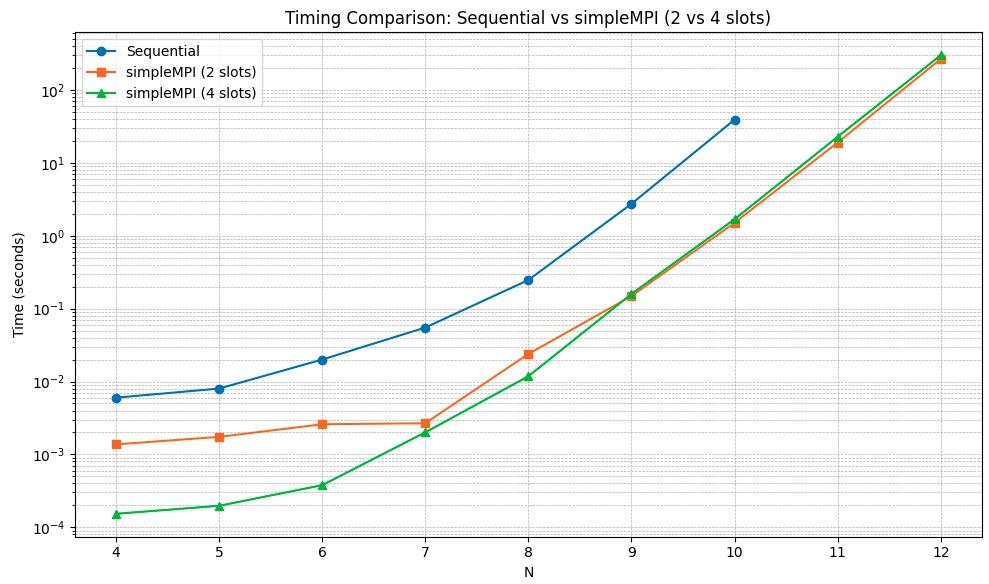
* For n=11:
  + **Sequential Code:** Crashes due to excessive memory.
  + **Parallel Code:** Completes successfully across all MPI processes.
* For n=12:
  + **Only 32 GB RAM Node** successfully completes execution.
  + **Other nodes** fail due to insufficient memory.

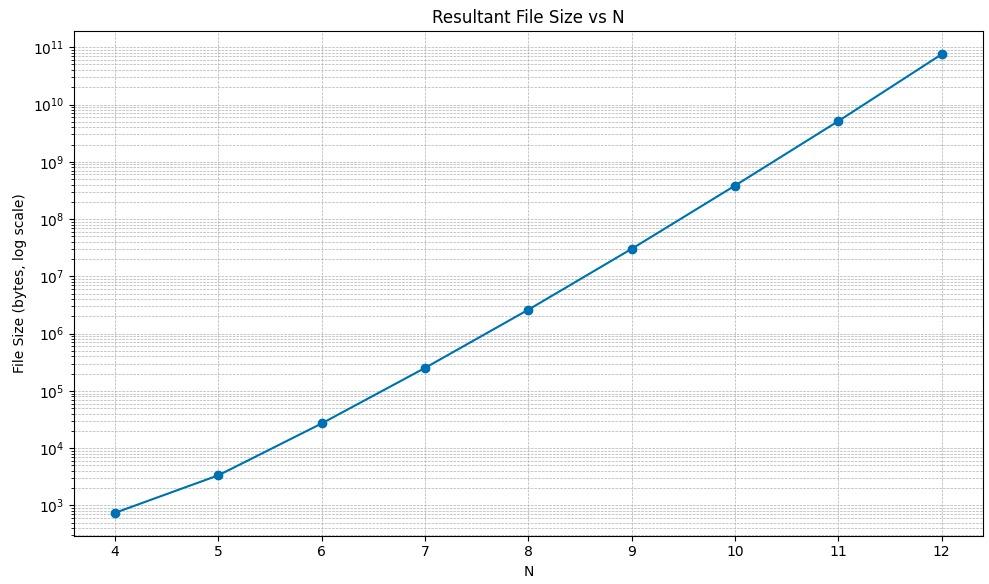
### **6.3 Chart: Execution Time vs N**

**

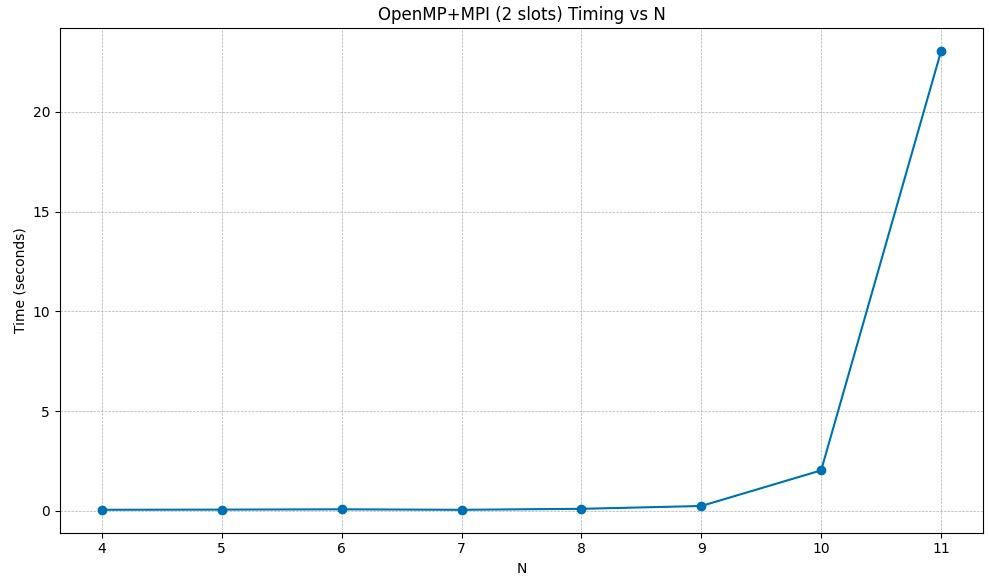
**

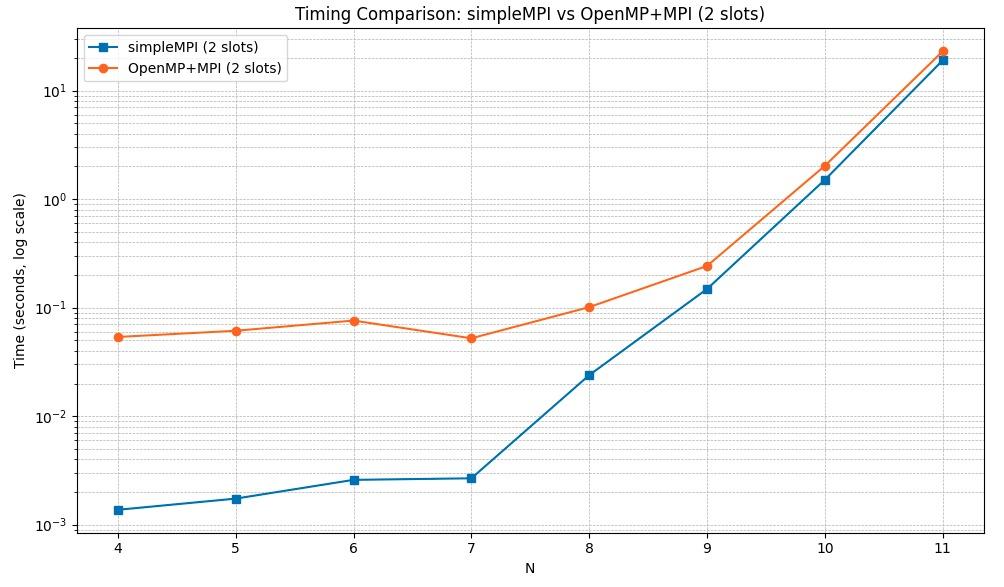


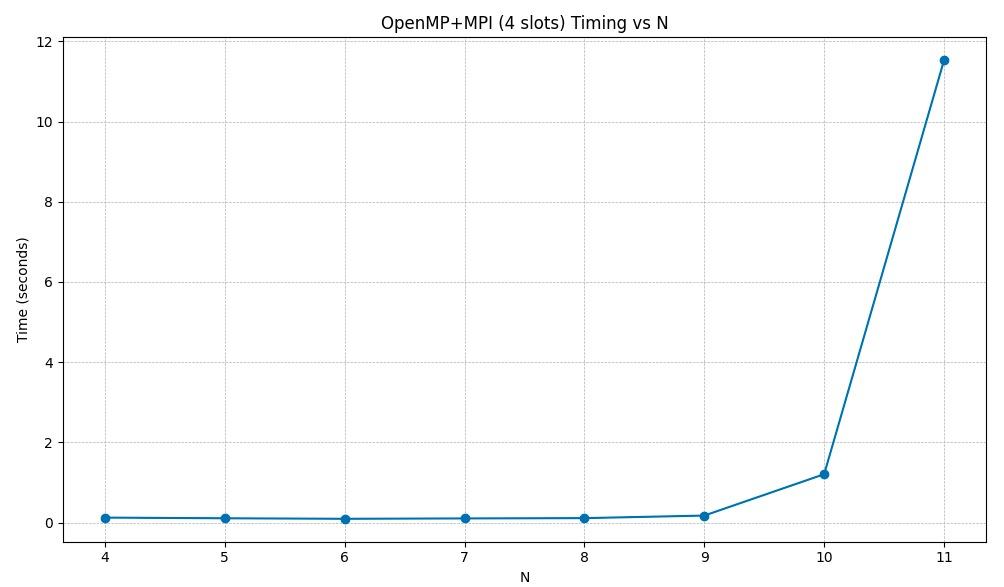


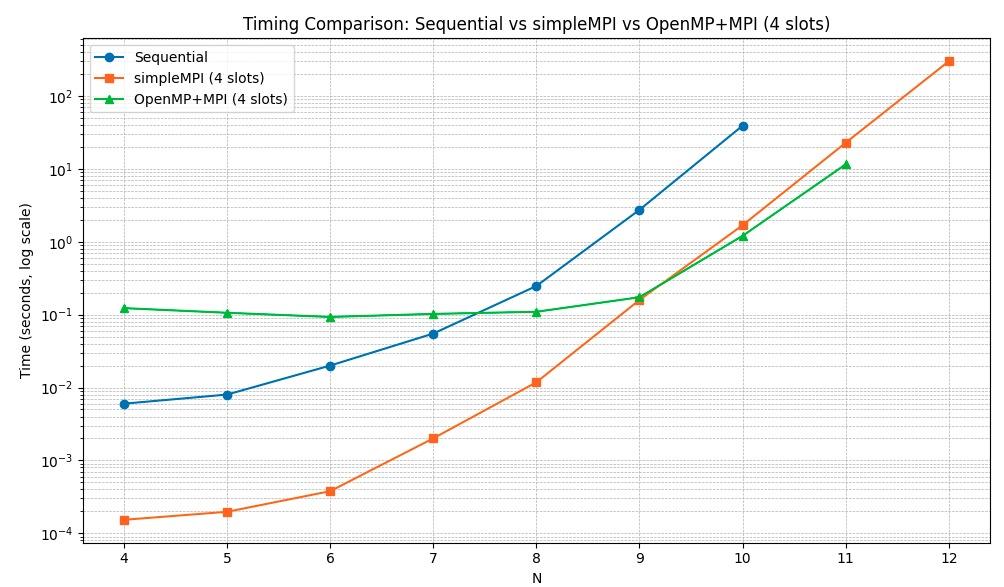


There were some storage issues Here.









It can be seen openMP starts doing better from N = 10

### **6.4 Chart: Speedup Comparison**

## 

## **7. Observations and Challenges**

* **High Setup Latency (20s)** observed on the MPI cluster due to networking overhead.
* **Homelab Performance** significantly better, attributed to single-node, high-bandwidth, and large shared memory.
* **I/O Bottlenecks** avoided using single-threaded final write after OpenMP region.

## **8. Conclusion**

This work demonstrates the benefits of hybrid parallelism in computing complex permutation structures. We implemented a memory-optimized, MPI+OpenMP solution to compute parent networks for Sn permutations. The parallel code scales well and enables processing of problem sizes that are infeasible with a sequential approach.

Future work may focus on:

* Reducing I/O bottlenecks using MPI-IO or distributed file systems.
* Implementing a memory-efficient compressed output format.
* Further optimizing inter-process communication using shared memory within nodes.

**Appendix**

### **A. Compilation and Execution Instructions**

mpicxx -O2 -fopenmp parallel\_mpi.cpp -o parallel\_mpi

mpirun --hostfile machines -np <num\_processes> ./parallel\_mpi <N>

### **B. Sample Output File Format**

vertex\_id,perm,T1,T2,...,T(n-1)

0,1-2-3-4,...,...