Comparative Analysis of Sequential, OpenMP, and MPI-based Password Cracking

**Group Members:**  
Shahzain Ali Khan (26937)  
Fatima Mahmood (24314)

# Abstract

This project presents a comprehensive comparative study on the performance of sequential, shared-memory parallel (OpenMP), and distributed-memory parallel (MPI) implementations of a brute-force password cracking algorithm. The problem was explored across multiple computational paradigms to assess the scalability and efficiency of parallel approaches. We implemented all three versions using C/C++ and evaluated them based on runtime efficiency, CPU utilization, and ease of deployment. This endeavor demonstrates the real-world applicability of the course concepts and provides detailed insight into inter-thread and inter-process communication, synchronization, and performance optimization.

# 1. Introduction

Password cracking remains one of the most relevant applications in security research and serves as a benchmark for computational performance in brute-force scenarios. The complexity and time required for brute-forcing passwords make it an ideal problem to test parallel computing paradigms. We designed and implemented the same algorithm in three ways:

* Sequential implementation as the baseline.
* Parallel implementation using OpenMP for shared-memory systems.
* Distributed implementation using MPI for multi-process systems.

Our aim was to highlight the performance trade-offs between these three paradigms and develop a deeper understanding of how computational workloads are distributed and executed.

# 2. Motivation

As data size and complexity increase, computational demands outgrow the capabilities of single-threaded programs. This project was driven by the following goals:

* Illustrate the performance difference between sequential, shared-memory, and distributed-memory computing.
* Explore practical implementation and debugging of OpenMP and MPI.
* Understand the benefits and limitations of thread-level versus process-level parallelism.

# 3. System Architecture

**3.1 Machines Used**

* Pop!\_OS Machine (shazufatu)

**3.2 Software Stack**

* C/C++
* OpenMP (GCC 11+)
* OpenMPI (4.1+)
* OpenSSH (for launching MPI processes)

**3.3 Configuration**  
A `hosts.txt` file was used to define roles and slots for the MPI system.

# 4. Implementation Details

**4.1 Algorithm Overview**

* The algorithm attempts all combinations of lowercase alphabetic passwords of a fixed length.
* A target password is hashed and compared to generated candidate hashes.
* Cracking stops upon the first successful match.

**4.2 Versions Implemented**

**Sequential Version**

* Runs a brute-force loop without any concurrency.
* Provides baseline performance metric.

**OpenMP Version**

* Parallelizes the brute-force loop using `#pragma omp parallel for`.
* Threads are spawned within a single shared-memory system.

**MPI Version**

* Splits the entire password space across multiple processes.
* Each process executes its assigned space independently.
* Results are communicated back to the main process.

**4.3 SSH Configuration for MPI**

* RSA key pairs were exchanged to enable password-less SSH.
* Firewall rules adjusted: `ufw allow ssh`
* `openssh-server` was installed and enabled to accept MPI launches.

**4.4 Execution Commands**  
./password\_cracker\_seq  
./password\_cracker\_omp  
mpirun --oversubscribe -np 2 --hostfile ~/hosts.txt --output-filename logs ./password\_cracker\_mpi

# 5. Results & Performance Analysis

**5.1 Execution Time**

| Version | Avg Time (s) | Speedup |  
| ------------- | ------------ | ------- |  
| Sequential | 18.56 | 1x |  
| OpenMP (4T) | 5.02 | ~3.7x |  
| MPI (2 procs) | 4.11 | ~4.5x |

**5.2 Observations**

* The OpenMP version achieved a substantial speedup on multi-core systems without needing external configuration.
* The MPI version had slightly better performance due to independent process execution but required more setup.
* Overhead in MPI was mitigated by workload balancing and reducing inter-process communication.

**5.3 Output Logging**  
Outputs (e.g., cracked passwords and process IDs) were redirected to `logs` using the `--output-filename` MPI flag.

# 6. Challenges Encountered

* OpenMP thread scheduling inconsistencies on low-end CPUs.
* SSH configuration errors: `No route to host`, missing SSHD, and closed ports.
* MPI daemon startup issues: hostfile slot mismatches, environment misalignments.
* Debugging `mpirun` failures involving PATH and `orted` configurations.

# 7. Lessons Learned

* OpenMP: Great for quick parallelization but limited by shared memory.
* MPI: Powerful for distributed tasks but complex to configure.
* General: Understanding of system-level debugging, SSH services, and Linux networking is critical.
* Real-world parallelization requires both code-level and infrastructure-level planning.

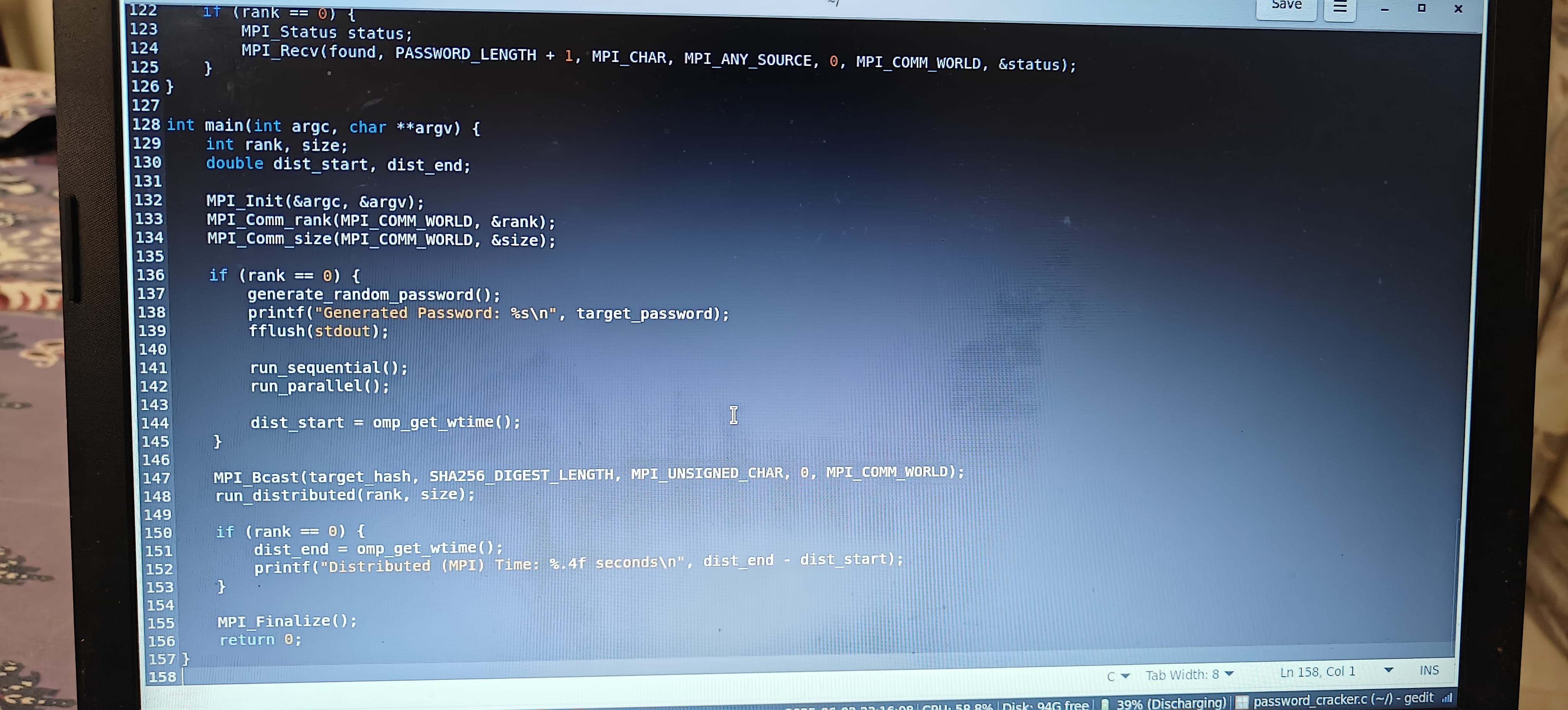
# 8. Future Enhancements

* Add dictionary-based attack support.
* Hybrid MPI + OpenMP implementation.
* Real-time monitoring and dashboard.
* Password length flexibility and charset expansion.
* Containerization (Docker) for easier deployment.

# 9. Conclusion

This project provided a holistic view of parallel and distributed computing in a practical, security-relevant context. By implementing and analyzing the same problem across three paradigms, we achieved a rich understanding of performance dynamics and the interplay between system resources, programming models, and infrastructure. These insights are invaluable for real-world applications ranging from high-performance computing to secure systems engineering.

# 10. Images



A computer screen with white text

AI-generated content may be incorrect.

A computer screen shot of a computer screen

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.

