A blue and white logo

Description automatically generated

**Semester Project**

**CS-3006 Parallel and Distributed Computing  
Name: Mannal Tariq & Muzammil Hassan**

**Roll no.: 21I-1378 & 21I-**

**Section: BS(CS) - B**

**Performance Analysis of Sequential and Parallel Log File Analysis**

**Q1: Parallel Detection of Malicious Activities in Large Log Files**

**Introduction**

This report evaluates the performance of log file analysis program using both a sequential approach and a parallel approach implemented with MPI in C language. The analysis investigates the speedup achieved by parallelization of the sequential program, as well as the impact of increasing the number of processes on execution time. For precise timing, we used clock() function from time.h library for the sequential version and MPI\_Wtime() from mpi.h library for the parallel version.

**Methodology**

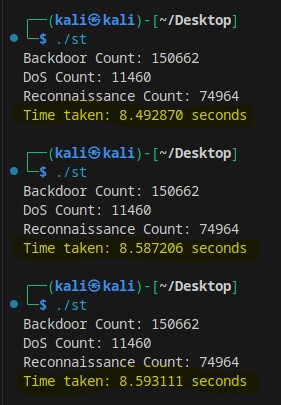
**Serial Execution Time Measurement**

The serial version of the log file analysis program was executed thrice to capture a range of performance metrics under consistent conditions. Each trial's execution time was measured using a timing function from the <time.h> library in C, ensuring accuracy. By conducting three independent trials, we aimed to account for potential variations in system performance and achieve an average execution time that reliably reflects the baseline performance of the serial implementation.

* **Trial 1 Execution Time**: 8.492870 seconds
* **Trial 2 Execution Time**: 8.587206 seconds
* **Trial 3 Execution Time**: 8.593111 seconds
* **Average Serial Execution Time**: 8.557729 seconds

**Snapshot of execution:**

Highlighted (yellow) is the time measured.

****

**Parallel Execution Time Measurement**

For the parallel version, the MPI\_Wtime function was employed to record the execution time across different numbers of processes. The program was run three times for each process configuration, and the times were averaged to minimize the impact of transient system load conditions.

* **Trial 1 Execution Time 4 Processes**: 1.869339 seconds
* **Trial 2 Execution Time 8 Processes**: 0.891299 seconds
* **Trial 3 Execution Time 16 Processes**: 0.713410 seconds

This procedure was repeated for different process counts, enabling a comprehensive comparison of the effect of increasing process counts on execution speed and efficiency.

**Snapshot of execution:**

**A screenshot of a computer program

Description automatically generated**Highlighted (yellow) is the time measured.

**Observed Speedup**

Using the average serial time (8.557729 seconds) as a baseline, the speedup achieved by increasing the number of processes was calculated as follows:

* 4 Processes:

Speedup= 8.557729 / 1.869339

Speedup ≈ 4.58

* 8 Processes:

Speedup = 8.557729 / 0.891299

Speedup ​≈ 9.60

* 16 Processes:

Speedup = 8.557729 / 0.713410 ​

Speedup ≈ 12.00

The results demonstrate that the parallel version significantly reduces execution time, achieving nearly linear speedup as more processes are added. However, the gains taper off, particularly when increasing from 8 to 16 processes, illustrating a common limitation in parallel computing.

**Analysis of Increasing Process Count**

**Performance Gains**

By adding more processes, we make these portions smaller, so the per process loads are smaller and execution overall is faster. For 4 processes, speedup was roughly 4.6x; 8 processes gave about 10x; 16 processes gave about 12x. The rise in this measurement is in line with Amdahl’s Law, that dictates that the maximum speedup is limited by the serial percentage of the task.

**Communication Overhead**

We observe that the speedup beyond 8 processes is reduced due to communication overhead and synchronization requirements. As more processes work together, they need to exchange data at boundaries to ensure consistent analysis across all parts of the log file. This communication takes time and becomes a larger factor as the number of processes grows, limiting the benefits of parallel processing.