**PDC Project Documentation**

Group Members:

Muhammad Daud Cheema 22i-0875

Mudassir Khalid 22i-1072

Muhammad Rahat Shafi 22i-1061

Project title: Butterfly computation

**Implementation Details:**

Sequential.cpp

This implementation serves as the baseline, performing all computations serially. It reads a bipartite graph from a file and stores it using adjacency lists. The program then iterates through each V-node to identify "wedges" (pairs of U-nodes connected to the same V-node) and checks for butterflies—substructures formed when two U-nodes share multiple V-node connections. The butterfly counting logic is entirely executed in a single thread without any parallelism, which makes it straightforward but slower on large datasets due to lack of concurrency.

OpenMP\_Parallel.cpp

In this version, OpenMP is used to introduce parallelism at the thread level. After reading and preparing the graph, the program converts the V\_to\_U map into a vector for indexed access and applies OpenMP directives to parallelize the loop that iterates through V-nodes. Each thread independently computes wedges and butterflies for its subset of nodes using #pragma omp for with dynamic scheduling. Shared variables (total\_wedges and total\_butterflies) are updated using atomic operations to ensure thread safety. This approach significantly speeds up the butterfly counting process on multicore systems while maintaining accuracy.

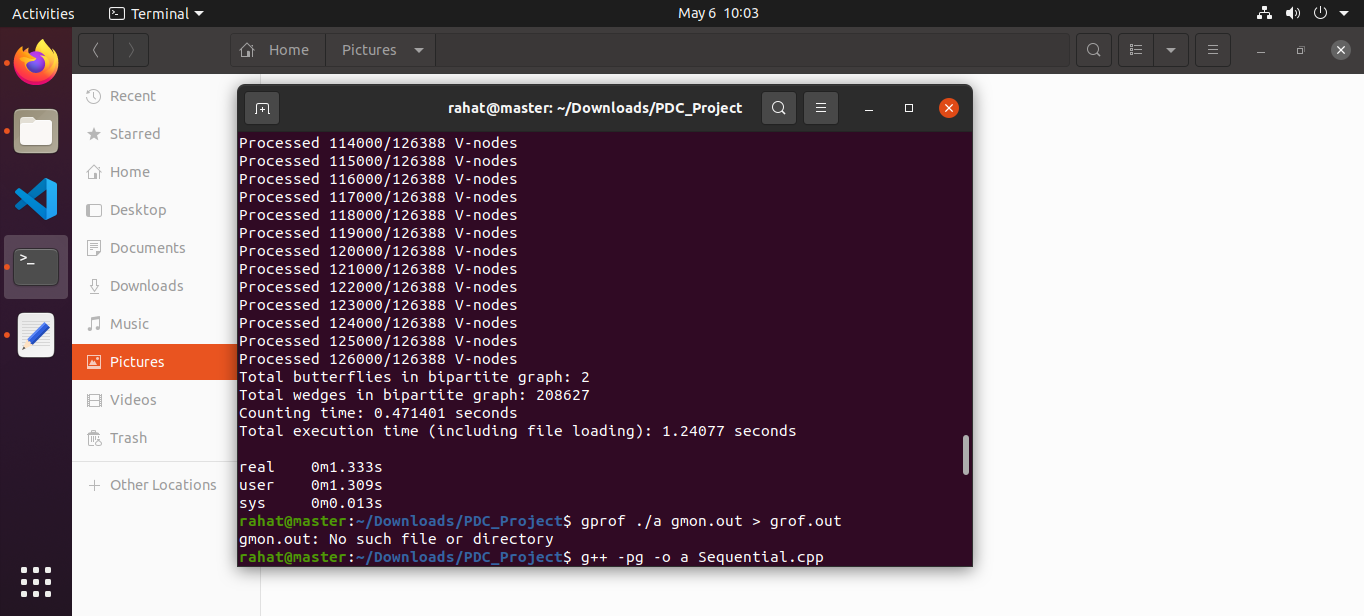
OpenMP\_MPI.cpp

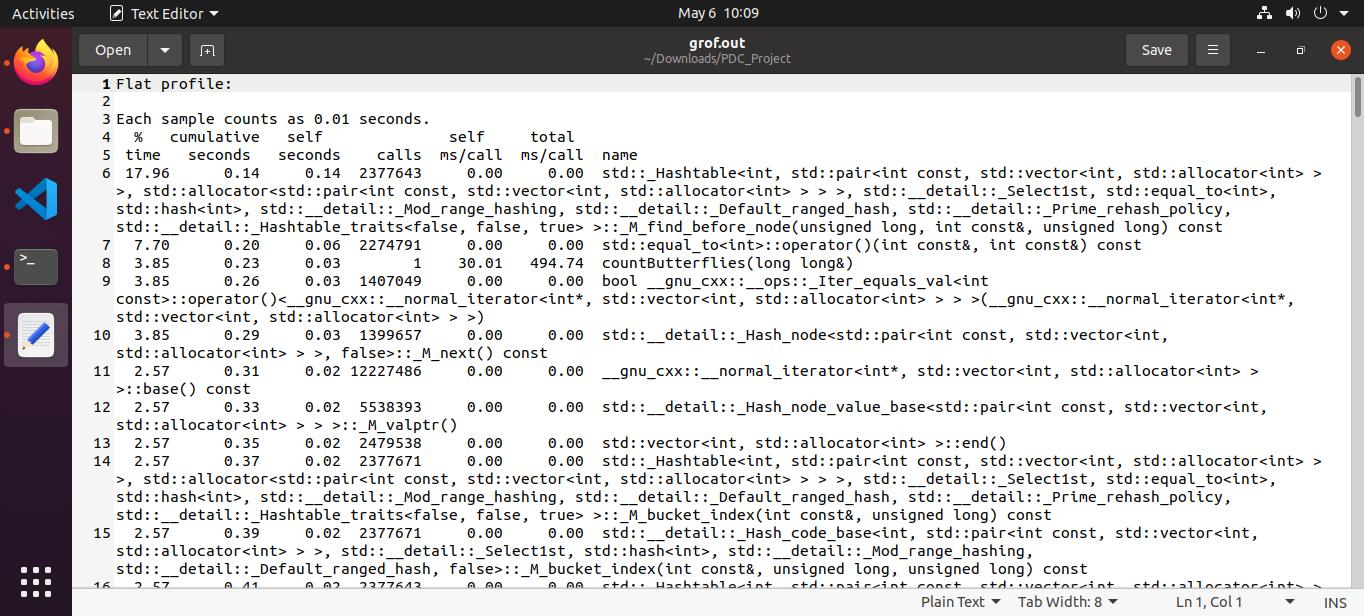
This hybrid implementation combines both OpenMP and MPI for parallelism across multiple nodes and cores. MPI is first used to initialize communication and determine the rank and size of each process. Each process reads the graph independently for simplicity (broadcasting is mentioned but not used). The list of V-nodes is divided among MPI processes, and each process uses OpenMP internally to parallelize the butterfly counting within its assigned chunk. After local computations, MPI\_Reduce gathers and aggregates results from all processes. This hybrid model efficiently utilizes distributed systems, enabling scalable performance improvements for large-scale graph data.

**Performance Analysis:**

Sequential Version:

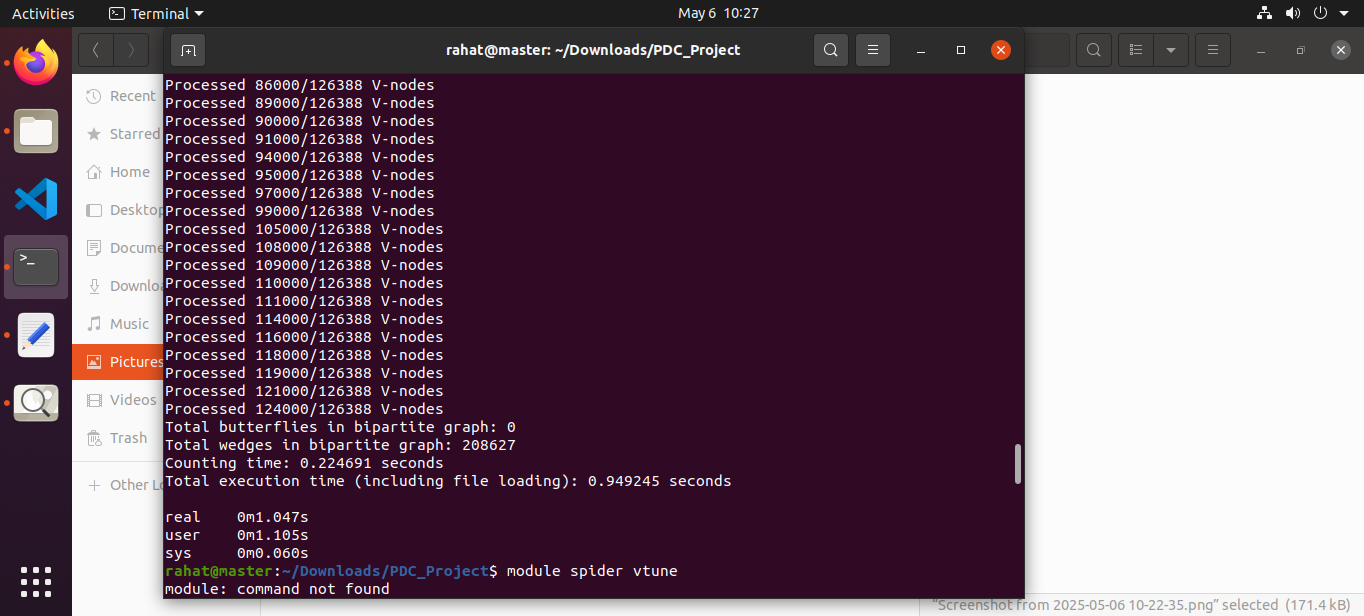
Time taken

Gprof Profiling



Parallel version using OpenMP

Time Taken:



Perf Profiling:

A screenshot of a computer

AI-generated content may be incorrect.