Parallel Depth-First Search (DFS) Algorithm: Project Report

1. Problem Statement and Baseline Design

Depth-First Search (DFS) is a traversal algorithm used extensively in applications such as simulations, web crawling, and network analysis. However, as graph sizes scale to millions of nodes and edges, a traditional serial DFS becomes computationally expensive and fails to exploit the full potential of modern multi-core architectures.

In this project, we begin with a baseline C++ implementation of DFS that processes a graph of 50,000 vertices using a single thread. Although the serial algorithm performs correctly, it exhibits limited scalability and increasing execution time with larger inputs. This limitation motivates our exploration of parallel DFS using OpenMP to leverage concurrency and improve performance.

2. Parallelization Approach and OpenMP Directives

The DFS algorithm's recursive nature provides opportunities for parallelization each recursive call that explores a node's neighbors can potentially be executed concurrently. Using OpenMP tasks, we parallelized the recursiv traversal to allow multiple branches of the search tree to execute simultaneusly.

OpenMP Directives Used

- #pragma omp parallel – creates a parallel region.
- #pragma omp single – ensures a single initial DFS launch.
- #pragma omp task shared(adj, visited, res) – spawns a parallel task for each neighbor.
- #pragma omp critical – ensures thread-safe access to shared data (e.g., visited array).

3. Performance Evaluation

Experimental Setup:

- Graph Size: 50,000 vertices
- System: Ubuntu via WSL
- CPU: (Insert your CPU model here)
- Compiler: g++ -fopenmp

Version	Threads	Time (ms)	Speedup	Efficiency
Serial	1	373.877	1.00×	100%
Parallel	16	125.015	2.99×	19%

The results show nearly 3× speedup on 16 threads. While this demonstrates tangible improvement, efficiency remains relatively low (≈19%), primarily due to synchronization overhead, uneven workload distribution, and task creation cost.

4. Amdahl’s and Gustafson’s Analysis

Amdahl’s Law quantifies the maximum speedup achievable given a serial fraction (fs) and number of processors

In our case, the serial portion includes critical sections, loop initialization, and the shared visited array. Even with 16 threads, these sequential components cap the maximum speedup.

Gustafson’s Law models how increasing the problem size can maintain efficiency.

As graph size grows, the relative cost of the serial part decreases, making parallel DFS more scalable. However, practical limitations—such as task overhead and cache contention—restrict ideal scaling.

5. Reflection and Remaining Bottlenecks

Despite improvements, several bottlenecks persist:
- Synchronization overhead: frequent entry into critical sections.
- Load imbalance: some threads finish early while others remain busy in deep recursion.
- Task creation overhead: fine-grained DFS recursion spawns too many lightweight tasks.
- Memory bandwidth: contention increases with high thread counts.