Parallel Processing Project 1 Report

Team Members: Reem Ooka & Nihar Lodaya

1) Architecture/Algorithm Used and Why?

- This DBSCAN implementation leverages several key C++ libraries to enhance performance and ensure thread safety. The `<iostream>` and `<fstream>` libraries handle input/output operations, reading points from a file and writing the clustering results back to an output file. Memory management is optimized with the `<memory>` library, using `std::unique_ptr` to handle dynamic memory allocation for neighbor arrays, ensuring memory is automatically deallocated to avoid leaks. The `<chrono>` library is used to measure the runtime, providing precise benchmarking of the algorithm's performance.

- Multi-threading is crucial for improving the efficiency of the DBSCAN algorithm, achieved through the `<thread>` library. The program divides the point dataset among multiple threads, allowing concurrent processing, which reduces the overall execution time for large datasets. Each thread processes its subset of points independently, while atomic operations (from the `<atomic>` library) ensure that shared data, like the `visited` and `clusterId` variables, remain thread-safe. This approach avoids the overhead of locking mechanisms such as mutexes, enabling faster and more efficient parallel execution.

- Overall, this implementation effectively balances performance and memory management, making it well-suited for large-scale datasets. The use of threads for parallelism, combined with dynamic memory management and thread-safe operations, ensures both speed and reliability.

2) Citations:

DBSCAN Algorithm: "DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is designed to discover clusters of arbitrary shape in spatial databases while handling noise. It requires minimal domain knowledge and can identify clusters based on density without predefined cluster counts." (Ester, Kriegel, Sander, & Xu, 1996).

Efficiency in Large Datasets: "Compared to other clustering algorithms like CLARANS, DBSCAN demonstrates superior efficiency, outperforming CLARANS by a factor of 100 in large spatial datasets." (Ester, Kriegel, Sander, & Xu, 1996).

Handling Noise in Clustering: "DBSCAN not only discovers clusters but also effectively identifies noise points, enhancing the quality of the clustering results, especially in real-world applications." (Ester, Kriegel, Sander, & Xu, 1996).

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), 226-231.

3) Table:

Threads	DBSCAN Parameters	Input size	Clusters	Noise Points	Runtime	Memory Consumption
1	15, 3	10	2	4	0.0010946s	0 KB
4	10, 3	100	11	16	0.0044513s	0 KB
8	12, 3	1000	6	42	0.0409652s	3964 KB
16	2.5, 2	10000	37	12	0.611885s	391523 KB

Testing Large Datasets -

When testing the code with larger datasets of 25,000 points, the program exited with the **error code -1073741571 (0xc00000fd)**. This error typically indicates a **stack overflow**, which occurs when the program exceeds the stack memory limit, often due to deep or excessive recursion or large local variables. To resolve this, consider reducing the depth of recursive calls, using iterative approaches, or increasing the stack size via compiler settings.

4) Explain the behavior of the data in the table and why?

1. **Single-Thread Performance:** With 1 thread, the DBSCAN algorithm processes points sequentially. For small input sizes, the runtime is fast, but as the dataset grows, performance degrades significantly. Memory consumption is minimal for smaller datasets.

2. **Scaling with Threads:** Increasing the number of threads improves runtime as the workload is divided. For example, with 16 threads on 10,000 points, the runtime drops significantly compared to single-thread performance. This parallelism is especially effective with larger datasets.

3. **Impact of DBSCAN Parameters:** DBSCAN parameters (epsilon, minPts) affect the number of clusters and noise points. Smaller epsilon values result in more

clusters and fewer noise points, while larger epsilon values lead to fewer, bigger clusters.

4. **Memory Consumption:** Memory usage grows with input size and more threads. For large datasets (10,000 points), memory consumption spikes due to dynamic memory allocation for neighbor lists and thread management.

5. **Runtime vs. Memory Trade-off:** Adding more threads improves runtime but results in diminishing returns and increased memory consumption due to synchronization and memory overhead.

5) Hardware Specs: **We used the Hancock Lab05 system to do our project.**

- Processor (CPU)
  13th Gen Intel(R) Core(TM) i7-13700, 2100 Mhz, 16 Core(s), 24 Logical Processor(s)
- RAM (Memory)
  Installed Physical Memory (RAM)16.0 GB
- BIOS
  BIOS Version/Date LENOVO S0IKT15A, 3/3/2023