Week 1 Report – Parallelism & Multithreading

# Reflection

During Week 1, I explored the fundamentals of parallel computing through a hands-on comparison of OpenMP and Pthreads for key parallel algorithms: matrix multiplication and prefix sum. The goal was not only to implement working versions, but also to measure performance, evaluate load balancing strategies, and analyze the trade-offs between ease of use and fine-grained control.

## What I Achieved

* Implemented and benchmarked matrix multiplication and prefix sum using:
  + Serial (baseline)
  + OpenMP with and without SIMD
  + Pthreads, including a row-wise optimized variant
  + Performed scalability testing from 4 to 8 threads on both small (128×128×128) and larger (512×512×512) matrices.
* Identified performance patterns across thread counts and analyzed when parallelization becomes advantageous vs when overhead dominates.

## Key Takeaways

* OpenMP is highly efficient for data-parallel workloads. Its simplicity and inbuilt compiler-level optimizations made it easy to achieve significant speedups
* Pthreads, while offering lower-level thread control, require careful task partitioning. The naive pthread version performed worse than serial due to overhead, but an improved row-wise implementation brought performance closer to OpenMP.
* Prefix Sum showed that the overhead of thread coordination on a short operation can outweigh the gains, especially for relatively small inputs.
* I learned to profile execution time, using thread scheduling.

## Challenges Faced

* On Windows, lack of native pthread support meant I had to use OnlineGDB to test pthread-based implementations.
* Ensuring thread safety while maximizing parallelism especially with shared memory like the result matrix required fixing race conditions.