Report

For this assignment, I focused on parallelizing four different serial programs using OpenMP directives. The main tasks I performed were:

1. Modifying the submit.sh Script: I adjusted the number of threads used by modifying the OMP\_NUM\_THREADS environment variable in the submit.sh file. This allowed me to experiment with different thread counts (1, 2, 4, 8, 16, and 32) to observe the impact on performance.
2. Adding OpenMP Directives: I examined the loops in each of the four problem files (problem1.cpp, problem2.cpp, problem3.cpp, and problem4.cpp) and added appropriate OpenMP pragmas to parallelize the code.

| **Threads** | **Problem 1** | **Problem 2** | **Problem 3** | **Problem 4** |
| --- | --- | --- | --- | --- |
| 1 | 0.42825 | 0.02647 | 0.07917 | 2.58295 |
| 2 | 0.32123 | 0.01410 | 0.04060 | 1.29247 |
| 4 | 0.18761 | 0.00837 | 0.02285 | 0.64666 |
| 8 | 0.12025 | 0.02662 | 0.03575 | 0.56853 |
| 16 | 0.11152 | 0.00824 | 0.02115 | 0.43478 |
| 32 | 0.09217 | 0.01039 | 0.02343 | 0.37678 |
| 64 | 0.18357 | 0.03174 | 0.04189 | 0.41322 |

## Analysis

* Problem 1: Shows consistent improvement with more threads, but a slight increase at 64 threads suggests diminishing returns.
* Problem 2: Significant gains up to a point, but some variability at higher thread counts may indicate overhead.
* Problem 3: Steady decrease in time, demonstrating good scalability.
* Problem 4: Most significant improvement, though slight increase at higher threads suggests optimal threading might be lower.

Overall, parallelization improved performance across all problems, with the most substantial gains in computationally intensive tasks like Problem 4.