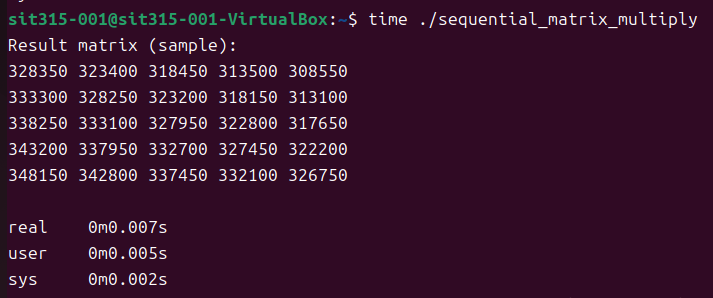
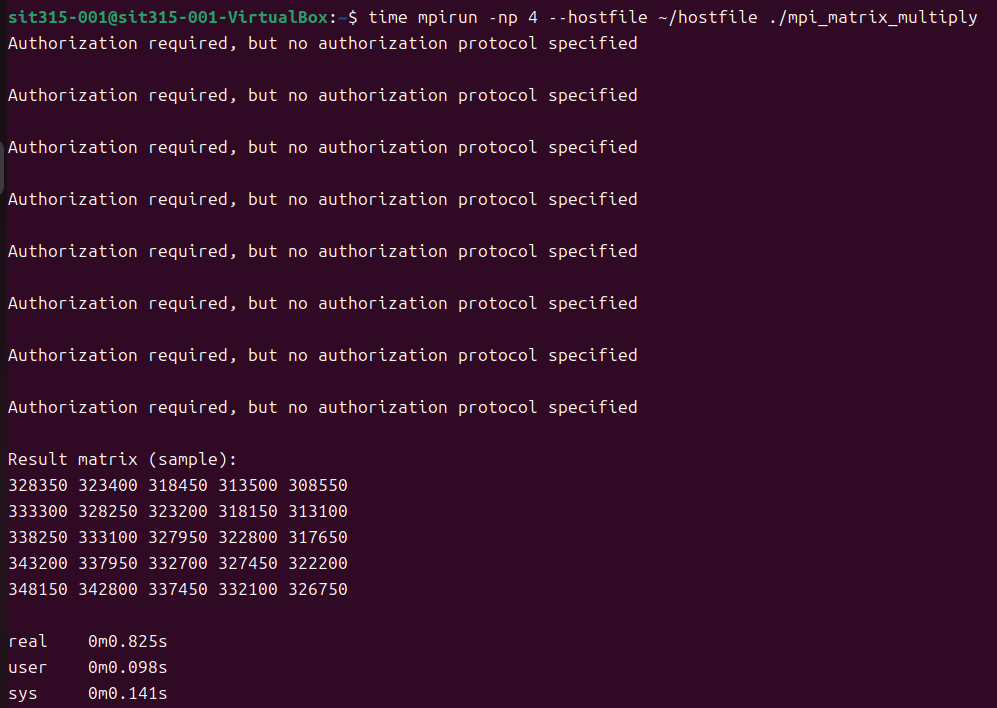
1.





**Time**

1. Sequential Version (sequential\_matrix\_multiply):

real: 0m0.007s

user: 0m0.005s

sys: 0m0.002s

2. MPI Version (mpi\_matrix\_multiply):

real: 0m0.825s

user: 0m0.098s

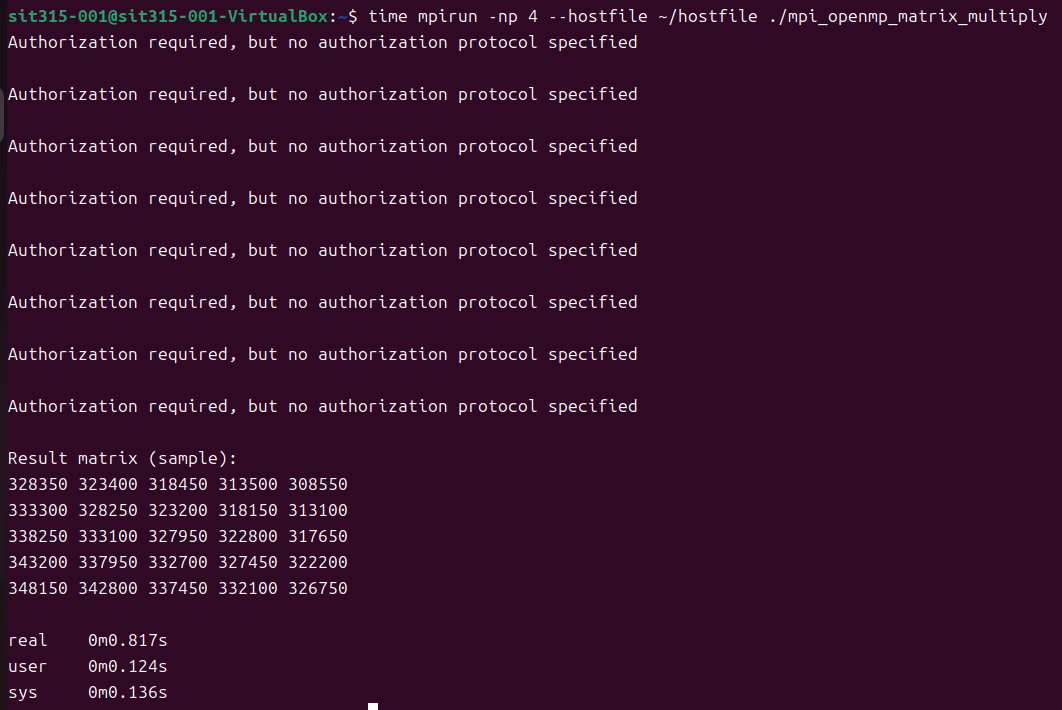
sys: 0m0.141s

**Conclusion**

Sequential Program: The sequential method is fast for small-scale operations due to minimal overhead. However, this method only uses a single CPU core and will not scale efficiently for larger matrices or more complex operations. Its real-time execution is the shortest, but it’s limited to small data sizes and lacks scalability.

MPI Programs: MPI introduces parallelization by distributing tasks across multiple processes, but this comes at the cost of inter-process communication overhead. While it makes sense for distributed systems with large data sizes, the observed real-time execution is much higher than the sequential approach due to this overhead.

2.



**Time**

3. MPI + OpenMP Version (mpi\_openmp\_matrix\_multiply):

real: 0m0.817s

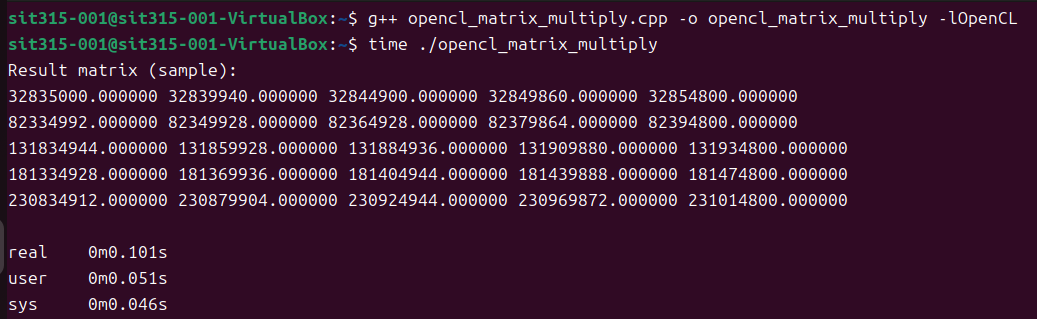
user: 0m0.124s

sys: 0m0.136s

**Conclusion**

MPI + OpenMP Program: Adds multi-threading but provides minimal additional benefit for small-scale tasks. The overhead from both MPI communication and thread management limits its effectiveness at small sizes, making it no better than MPI alone for smaller problems.

3.



**Time**

real: 0m0.101s

user: 0m0.051s

sys: 0m0.046s

**Conclusion**

OpenCL Program: OpenCL shows a dramatic reduction in execution time compared to the other methods. Leveraging GPU acceleration for parallel computation of matrix elements results in a significant performance gain. The real-time execution is the lowest of all methods, highlighting OpenCL’s suitability for matrix operations where the workload can be parallelized efficiently across a large number of GPU threads.

Github Link

Video Link