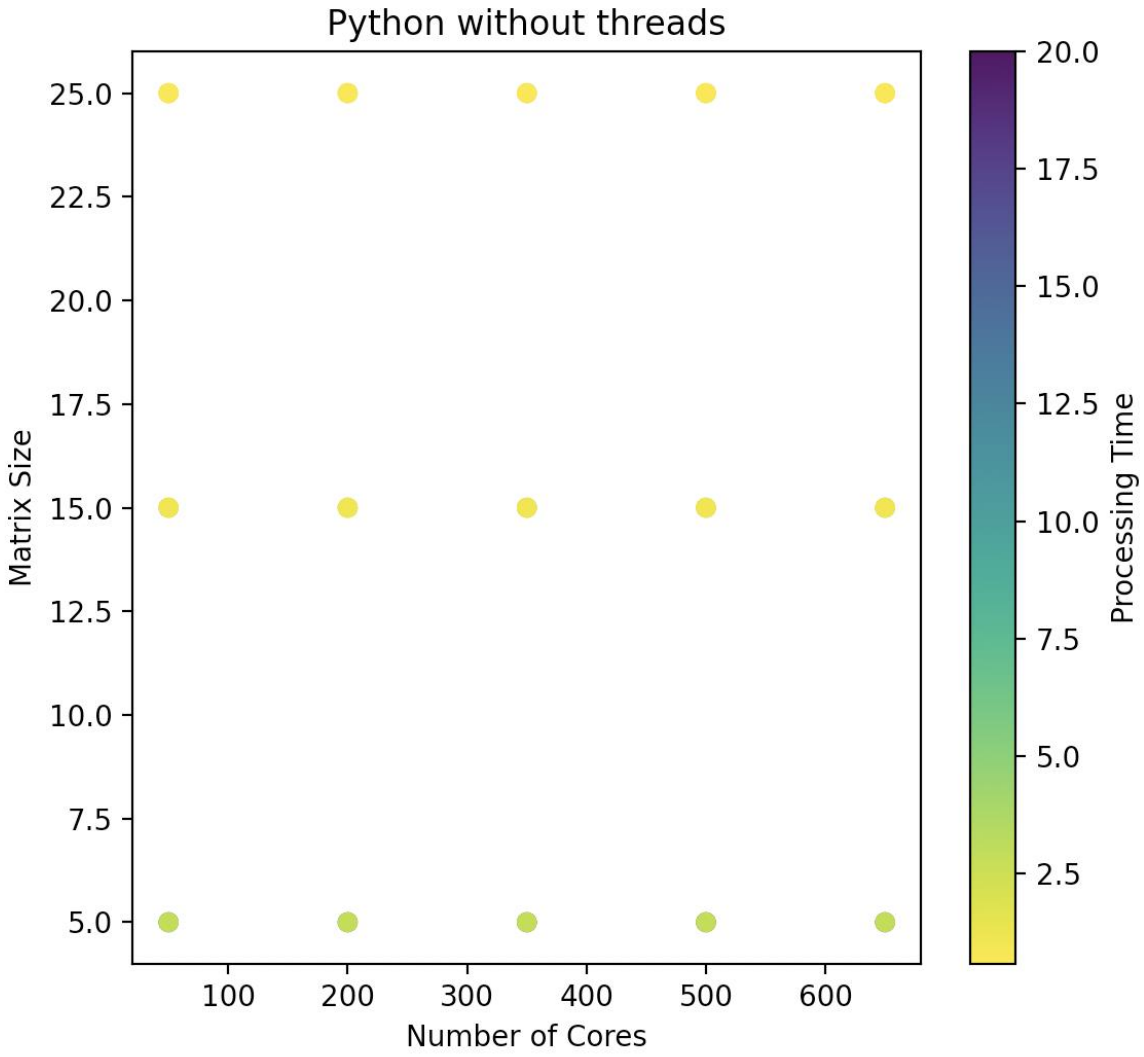
Parallelization Strategies for Optimizing Matrix Multiplication Performance

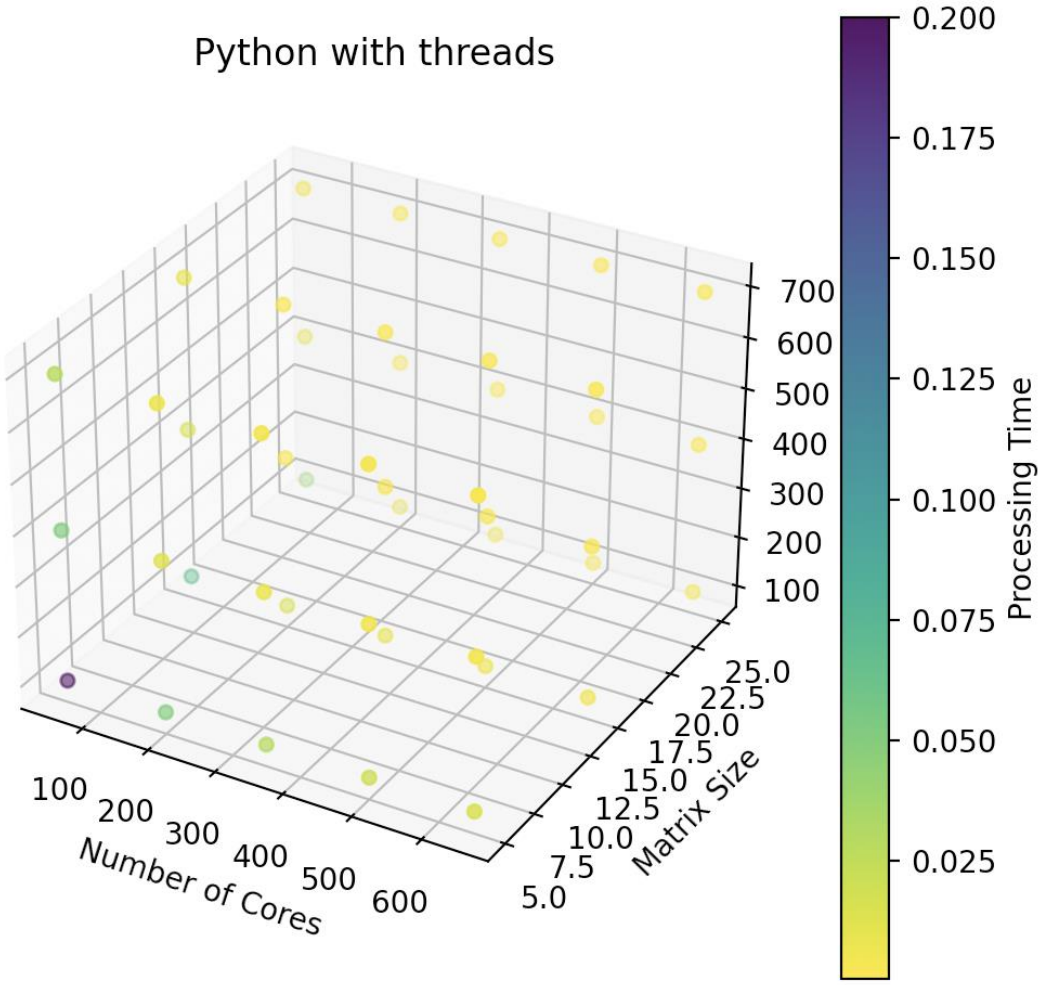
Python without Threads:

The Python program without threading shows the highest processing times among all the programs implemented. The processing time increases as the number processes, and matrix size increase.



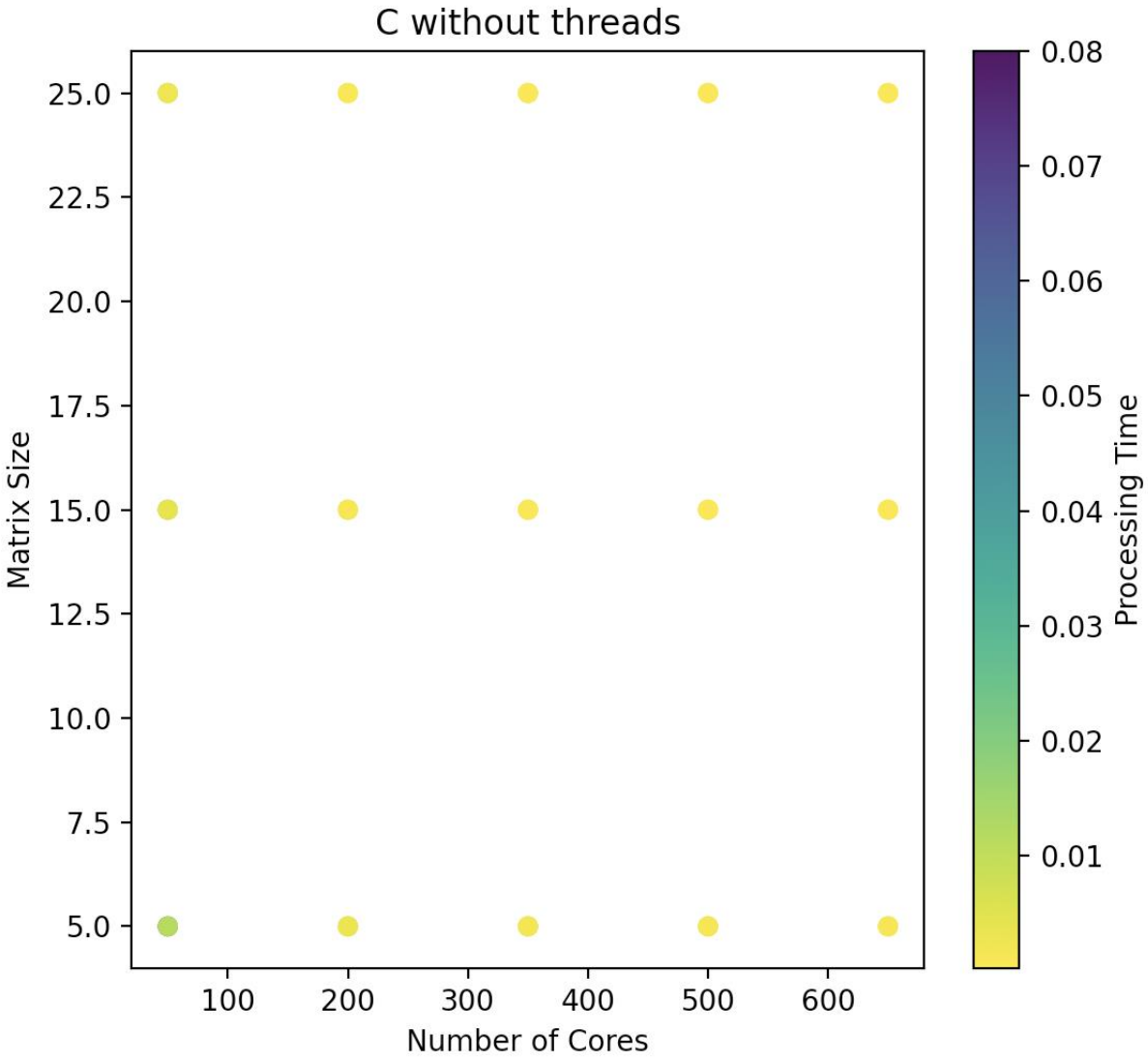
Python with Threads:

The Python program with threading support exhibits better performance compared to the non-threaded version, as indicated by the lower processing times. However, the improvement is still limited due to the GIL's impact on CPU-bound tasks (source: Claude). The processing time decreases with increasing threads, processes, and matrix size, but not as significantly as the C programs.



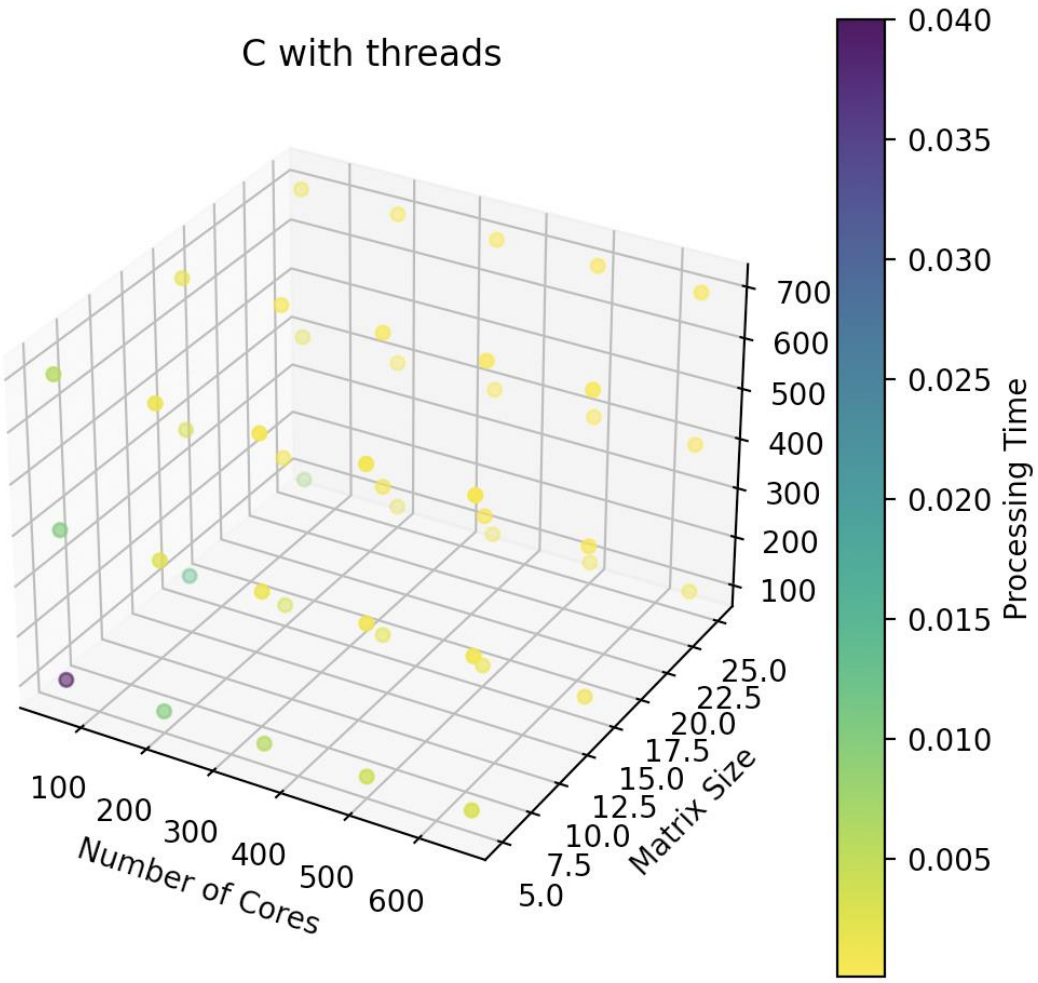
C without Threads:

The non-threaded C program performs better than both Python programs, with processing times approximately significantly lower than the Python without threads version. The processing time decreases as the number of threads, processes, and matrix size increase, but the improvement is limited by the lack of parallel execution.



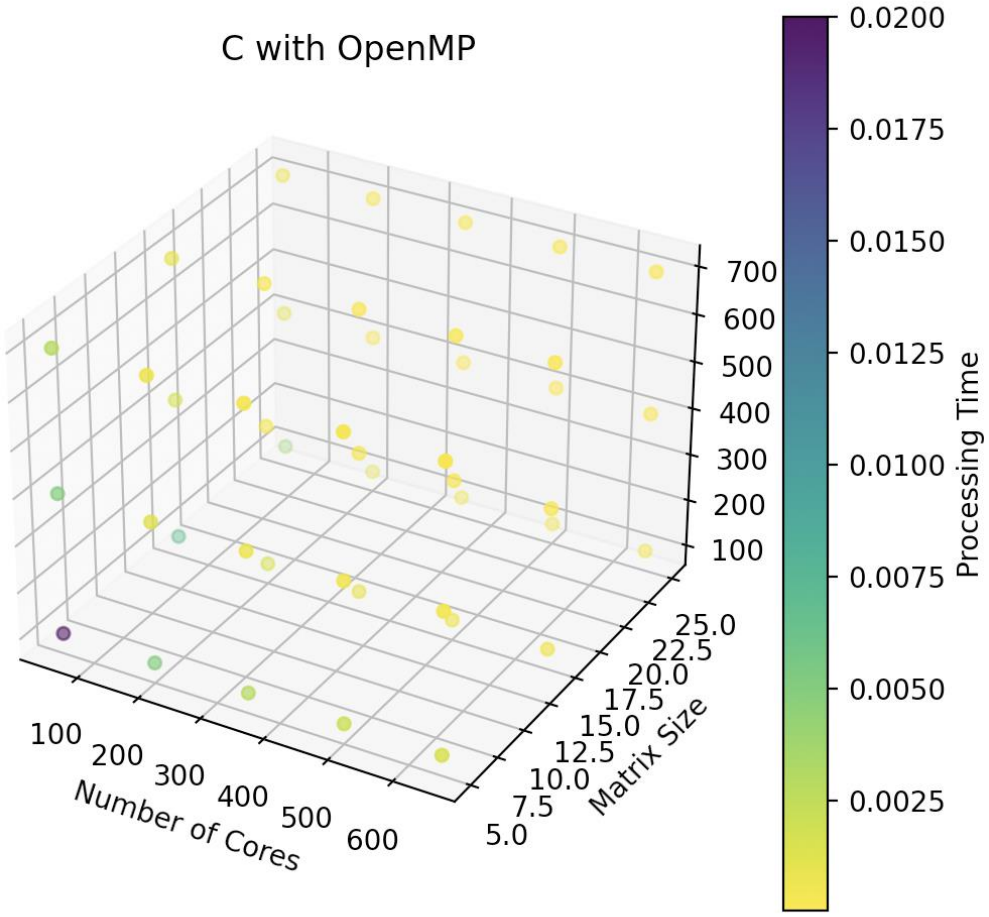
C with Threads:

The C program with threading support shows a significant performance improvement over the non-threaded C version. The processing times are approximately half of the non-threaded C program, and they continue to decrease as the number of threads, processes, and matrix size increase. This improvement is due to the effective utilization of multiple threads for parallel execution of the matrix multiplication task.



C with OpenMP:

The C program with OpenMP support exhibits the best performance among all the programs tested. The processing times are the lowest, approximately half of the threaded C program, and continue to decrease as the number of threads, processes, and matrix size increase. OpenMP enables efficient parallelization of the matrix multiplication task by automatically distributing the work across available threads and cores, leading to the best performance,



In summary, the C program with OpenMP support demonstrates the most efficient performance for matrix multiplication, followed by the threaded C program, non-threaded C program, threaded Python program, and non-threaded Python program. The OpenMP and threading implementations in C effectively utilize the available hardware resources (threads and cores) to parallelize the matrix multiplication task, resulting in significant performance improvements as the problem size (matrix size) increases.