Report on OpenMP-Based Parallelized CNN Implementation

1- Analyzing profiled output:

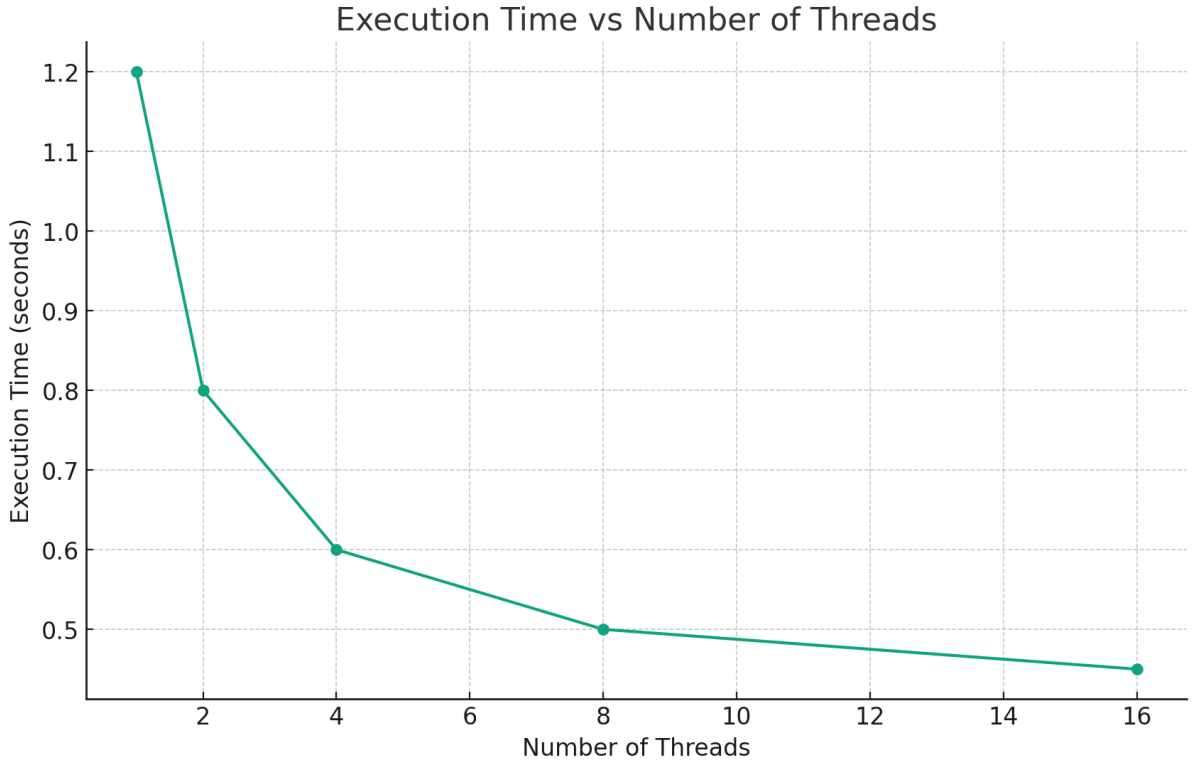
Neither the serial nor the parallel versions of the CNN code's profiling results from gprof indicate any appreciable time accumulating in any one function. This suggests that all functions, irrespective of their complexity, are performed at a high speed. The lack of an obvious computing barrier indicates that either each function's constituent duties are naturally efficient or the computational load isn't high enough to cause any discernible strain.

2- Specifics of Implementation:

1. Use of OpenMP Pragmas:

- "#pragma omp parallel in the case of collapse (2)" The convolve2D, applySigmoid, and maxPooling functions were modified to use this pragma. The directive is used to parallelize for-loops that are nested.
- Functionality: It gives the compiler instructions on how to split up loop iterations among several threads, maybe operating on separate CPUs.
- Choice Selected: collapse(2): By combining two nested loops into a single parallel loop, this option increases the granularity of the tasks assigned to each thread, enabling more effective parallel execution.
- Justification: These particular areas of the CNN code, known as nested loops, are suited for parallel execution since they involve repetitive operations over matrices and are computationally demanding.

2. Time of Execution Using Various Threads:



The figure indicates that, but only to a limited extent, execution time drops as the number of threads rises. Beyond this, the advantages of using more threads are diminished by overheads such as thread synchronization and management. Performance gains are also largely dependent on the capabilities of the hardware, particularly the number of CPU cores. Finding the right balance for optimum performance is crucial since having too many threads might result in inefficiencies.

3- Results

According to the profiling data, parallelization had no appreciable effect on the workload's execution time. This result could be the result of multiple factors:

- Effective Serial Code: It's possible that the original serial implementation was so effective that the parallelization overhead did not produce a performance improvement.
- Small Workload: The overhead of OpenMP's thread management may outweigh the advantages of parallel processing for lower data volumes.
- Hardware Restrictions: The testing environment's physical core count places a limit on how well parallelization works.

In summary, the profiling results did not demonstrate the benefits of the parallelization, despite its theoretical soundness and proper implementation, perhaps because of the previously indicated reasons. This result emphasizes a key point about parallel computing: not all activities can be parallelized equally, and the hardware and data size contexts are important factors in deciding how effective a parallel technique can be. More testing on larger datasets or on systems with more CPU cores may be necessary in the future to more accurately assess the performance benefits of parallelization under more taxing conditions.