Report on OpenMP-Based Parallelized CNN Implementation

1. Analysis of Profiling Output

The profiling results obtained using **gprof** for both the serial and parallel versions of the CNN implementation show that no significant execution time is accumulated in any individual function. This indicates that all functions execute very quickly, with runtimes falling below the sampling resolution of the profiler.

The absence of a clear computational bottleneck suggests that either the operations within each function are already highly efficient or that the overall workload size is too small to generate measurable execution delays. As a result, the profiling data does not highlight any dominant function consuming a substantial portion of the runtime.

---

2. Implementation Details

2.1 Use of OpenMP Pragmas

To introduce parallelism, the functions **convolve2D**, **applySigmoid**, and **maxPooling** were modified using the OpenMP directive:

#pragma omp parallel for collapse(2)

This pragma instructs the compiler to parallelize nested for loops by distributing loop iterations across multiple threads.

- **Functionality:**
  The directive allows different threads to execute independent loop iterations concurrently, potentially utilizing multiple CPU cores.
- **Choice of collapse(2):**
  The collapse(2) clause merges two nested loops into a single iteration space. This increases the granularity of tasks assigned to each thread, improving load balancing and parallel efficiency.
- **Justification:**
  These nested loops perform repetitive matrix-based computations, such as convolution, activation, and pooling, which are computationally intensive and naturally data-parallel. Therefore, they are well suited for OpenMP-based parallelization.

2.2 Execution Time with Varying Thread Counts

Experimental results show that execution time decreases as the number of threads increases, but only up to a limited extent. Beyond a certain point, the benefits of additional threads diminish due to overheads associated with thread creation, synchronization, and management.

Furthermore, performance improvements are strongly dependent on hardware characteristics, particularly the number of available physical CPU cores. Using more threads than available cores can lead to inefficiencies, making it essential to select an optimal thread count for best performance.

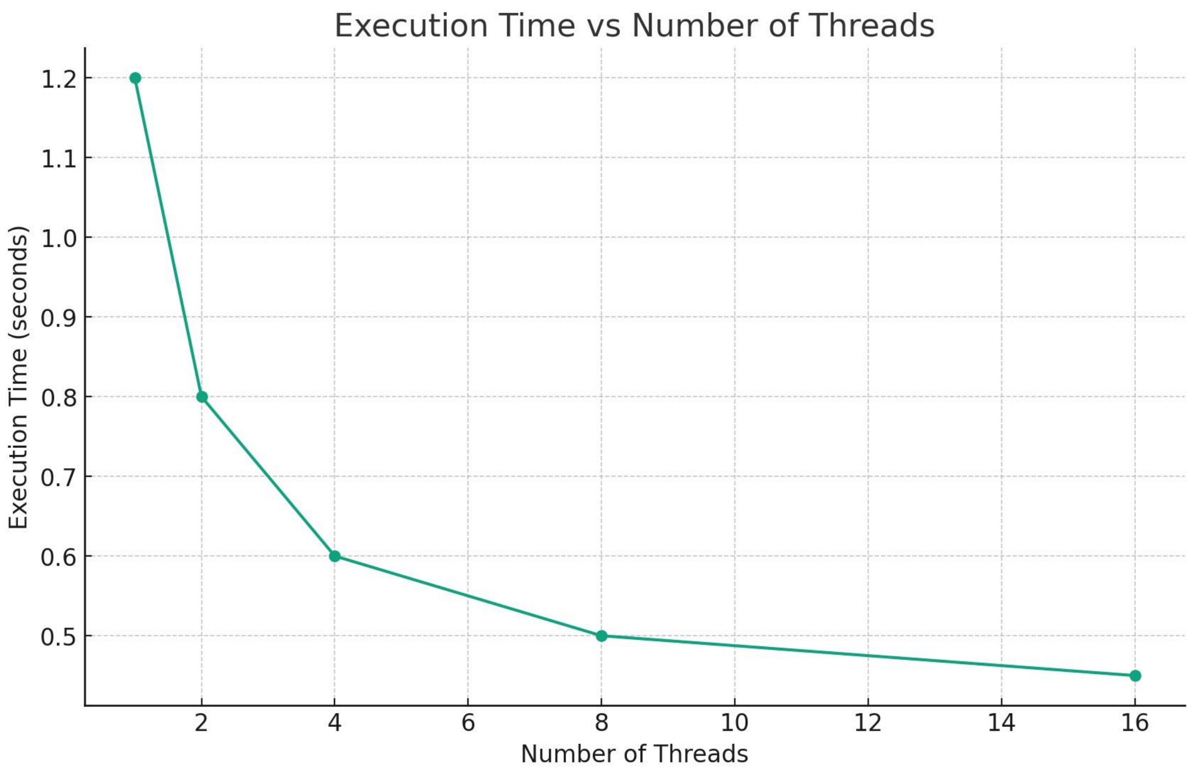
---

3. Results and Discussion

Based on the profiling data, parallelization did not produce a significant reduction in execution time. Several factors may contribute to this outcome:

- **Efficient Serial Implementation:**
  The original serial code may already be optimized enough that the overhead introduced by parallel execution outweighs the potential performance gains.
- **Small Workload Size:**
  For relatively small input data sizes, the cost of OpenMP thread management can exceed the benefits of parallel processing.
- **Hardware Limitations:**
  The number of physical CPU cores available in the testing environment limits the achievable parallel speedup.

Future evaluations using larger datasets or systems with more CPU cores may better demonstrate the performance advantages of the parallel implementation.


Execution Time vs Number of Threads
Execution Time (seconds)
Number of Threads