 # Prerequisites
GCC compiler with OpenMP support
Terminal for command-line execution
 # Compilation
To compile each program, navigate to the source code folder and run:

 # For serial programs:
g++ SerialProgramName.cpp -o OutputFileName

 # For parallel programs:
g++ -fopenmp ParallelProgramName.cpp -o OutputFileName

 # Execution
./OutputFileName

Description and Performance Analysis

Summing an Array
Serial: The serial version employs a simple loop for summing an array.
Parallel: Uses OpenMP to parallelize the loop.
Performance: Parallel version is faster for larger arrays.

LU Decomposition
Serial: Utilises standard LU Decomposition algorithm.
Parallel: Uses OpenMP to parallelize the main loop.
Performance: Parallel version surprisingly performs slower, likely due to overhead and imperfect parallelizability.

Matrix Inversion
Serial: Uses Gauss-Jordan Elimination for matrix inversion.
Parallel: Employs OpenMP to parallelize determinant calculation and row operations.
Performance: Parallel version might not show expected speedup due to data dependencies and thread overhead.

Results
Performance analysis indicates that while parallelization has its benefits, it doesn't universally offer speedup for all problem types. The choice between serial and parallel should be based on problem complexity, size, and available resources.

Summing an Array
Serial: The time complexity for the serial version is O(N), where N is the size of the array.
Parallel: Despite parallelization, there's overhead in thread creation and synchronisation, which offsets the speedup for smaller arrays.

Comparison: Parallel version performs better for larger arrays. For smaller arrays, the serial version outperforms due to less overhead.

LU Decomposition
Serial: The time complexity is O(N^3) for an NxN matrix.
Parallel: In theory, parallelization should reduce time, but due to overhead and imperfect parallelizability, the result is not as expected.
Comparison: Surprisingly, the parallel version took more time than the serial one. This could be due to cache coherence issues, thread overhead, or imperfect parallelization of the algorithm.

Matrix Inversion
Serial: This task has a high computational cost, especially when calculating determinants, which again has a time complexity of O(N!).
Parallel: Even with parallelization, there is a challenge due to data dependencies and thread management overhead.
Comparison: Similar to LU Decomposition, the parallel version doesn't offer a significant speed-up and may perform worse for smaller matrices.

Summary
In general, parallel programming shows its strength for computationally intensive tasks that are "embarrassingly parallel." However, due to algorithmic complexity, overhead, and other issues like cache coherence, parallel versions of some problems like LU Decomposition and Matrix Inversion might not show expected speedup compared to their serial counterparts. Therefore, the choice between parallel and serial approaches should be made based on the problem size, nature, and the resources available.