Lab 6: Performance Analysis of Parallel Matrix Multiplication

Objective:
The primary aim of this experiment is to investigate and quantify the performance enhancements achieved through parallelization in matrix multiplication. Matrix multiplication, a fundamental operation in numerous scientific and engineering applications, can be computationally intensive, especially with large matrices. By employing parallel computing techniques, we aim to improve the efficiency and speed of this operation.

Parallelization Strategies:
In this experiment, we explore two main parallelization strategies:

Single Instruction, Multiple Data (SIMD): This approach involves using ARM's neon instructions to perform multiple operations simultaneously. SIMD is particularly relevant for operations like matrix multiplication, where the same operation (multiplication and addition) is repeated across many data elements.

Multithreading with OpenMP: This technique leverages the power of multiple CPU cores to execute several parts of the matrix multiplication process concurrently. By dividing the task among multiple threads, we can significantly reduce the overall computation time.

matmul	arm.csc.calpoly.edu
Runtime SIMD	
Runtime 1 Thread	**Run-Time: 65.361 s**
Runtime 2 Threads	**Run-Time: 41.955 s**
Runtime 4 Threads	**Run-Time: 24.186 s**
Runtime 8 Threads	**Run-Time: 15.334 s**
Runtime 16 Threads	**Run-Time: 10.248 s**
Runtime 32 Threads	**Run-Time: 7.787 s**
Runtime 64 Threads	**Run-Time: 6.685 s**

matmul	arm.csc.calpoly.edu
Runtime OpenMP	
Runtime 1 Thread	**Run-Time: 66.249 s**
Runtime 2 Threads	**Run-Time: 41.738 s**
Runtime 4 Threads	**Run-Time: 25.426 s**
Runtime 8 Threads	**Run-Time: 15.825 s**
Runtime 16 Threads	**Run-Time: 11.857 s**
Runtime 32 Threads	**Run-Time: 9.304 s**
Runtime 64 Threads	**Run-Time: 8.132 s**

Performance Measurements:

Objective: To comprehensively evaluate the performance of our SIMD and OpenMP implementations, we focus not only on execution time but also on cache efficiency.

Measuring Execution Time and Cache Metrics:

Tool Used: We employed the time command on a Unix-like system to measure the execution time and cache performance.

Command Syntax: The command time ./mm > myout was used, where ./mm represents the executable of our matrix multiplication program.

Output Analysis: This command generates an output file, myout, containing the details of execution time along with system resource usage. Cache Performance

Metrics: Specifically, we measure the number of cache misses and cache references. These metrics provide insights into how effectively our program utilizes the CPU cache.

Importance of Cache Metrics:

Cache Misses: A high number of cache misses can significantly slow down program execution, as the processor needs to fetch data from slower main memory.

Cache References: Indicates how often the program accesses the cache. A higher number of references with fewer misses implies efficient cache usage.

Conclusion:
Our Lab 6 experiment distinctly demonstrates the efficacy of parallel computing techniques in optimizing matrix multiplication, a critical operation in various scientific and engineering tasks. Employing SIMD and OpenMP strategies led to significant reductions in execution time, with the runtime decreasing from over 65 seconds to less than 7 seconds using SIMD, and from around 66 seconds to just over 8 seconds using OpenMP as the number of threads increased. This marked improvement, coupled with insights into cache performance, underscores the potential of parallelization in enhancing computational efficiency. The results not only confirm the advantages of using multiple cores and SIMD instructions for repetitive computations but also pave the way for similar applications in other computationally intensive processes.