# 2024.1 Multicore Computing, Project #1

# Problem 2

## Introduction

This report provides a performance analysis of a parallelized matrix multiplication algorithm implemented in Java. The execution time of the algorithm was measured while varying the number of threads used, demonstrating the impact of parallel computing on performance.

## Methodology

The matrix multiplication was carried out using a block decomposition approach. This method involves partitioning the first input matrix into horizontal strips, which are then processed in parallel. Each thread was responsible for computing the result of its assigned strip multiplied by the second input matrix.

## System Specification

Processor: Apple M1 Pro

Core Count: 10 cores (8 performance cores, 2 efficiency cores)

Clock Speed: Approximately 3.2 GHz

RAM: 16GB of unified memory

GPU: Integrated 16-core GPU

Neural Engine: 16-core, designed for advanced machine learning tasks

Memory Bandwidth: Up to 200GB/s

The Apple M1 Pro's architecture is particularly well-suited for parallel processing due to its balance of performance and efficiency cores.

# Execution Details

## Decomposition Method: Block Decomposition

The matrix multiplication was executed using the block decomposition method, where the workload was divided into horizontal blocks, each processed by a separate thread. This choice was predicated on several key benefits that align well with the characteristics of the computation and the specifics of the hardware:

**Cache Efficiency:**
Leveraging spatial locality, block decomposition ensures that the data processed by each thread is contiguous in memory, leading to better cache usage and fewer cache misses.

**Reduced Synchronization Overhead:**
This method inherently requires less synchronization between threads, as each one works on a separate block, reducing the potential for data contention and locking overhead.

**Balanced Workload:**
Work is evenly distributed among threads, with each handling an approximately equal number of rows from the matrix. This balance allows for effective use of the CPU's multi-core architecture.

**Simplicity and Scalability:**
Block decomposition is straightforward to implement and scales well with additional threads, which is particularly beneficial on hardware with multiple cores and high memory bandwidth.

## Task Size

The size of the task allocated to each thread was dynamically adjusted based on the number of available threads, ensuring that all threads remained actively engaged in computation and that workload distribution remained as uniform as possible.

## Threads Used

The performance of the algorithm was tested using different thread counts: 1, 2, 4, 6, 8, 10, 12, 14, 16, and 32. This variation was intended to identify the optimal number of threads that maximizes performance on the Apple M1 Pro processor, which has 8 performance cores. The execution times captured at each thread count level were analyzed to determine the point of diminishing returns, where the overhead of managing additional threads outweighs the benefit of parallel execution.

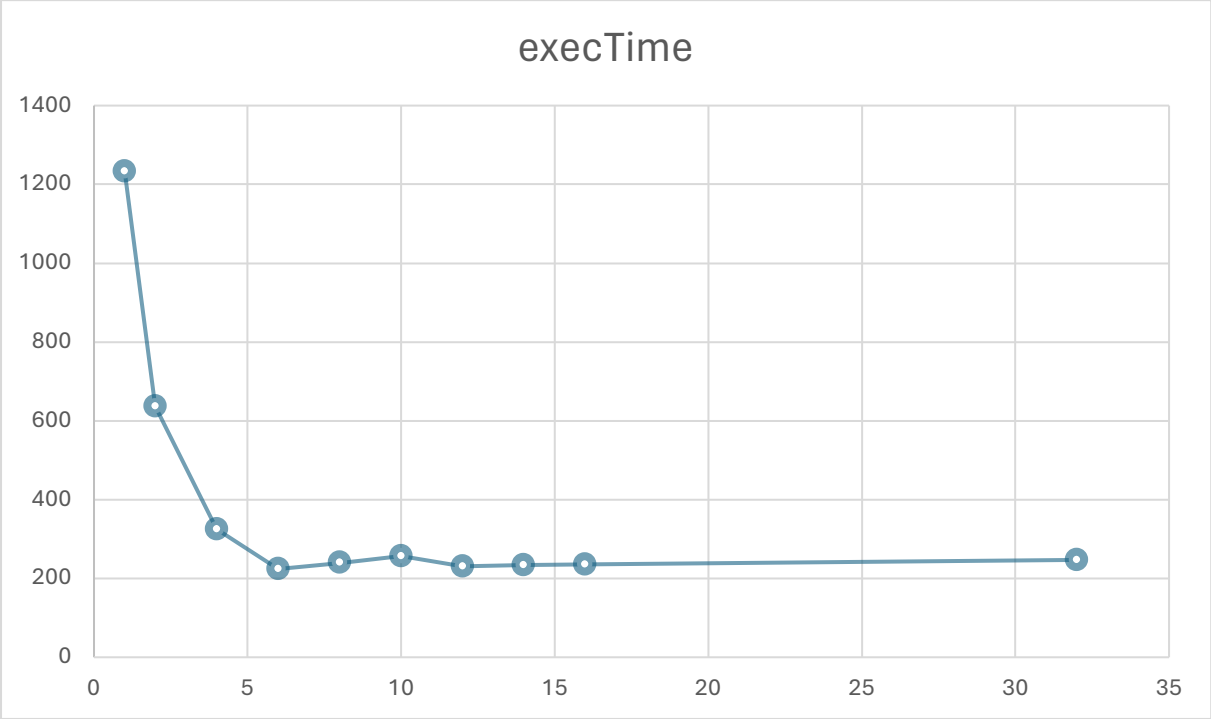
Through this methodical approach, we discerned that utilizing 6 threads corresponds to the best performance, leveraging the high-efficiency cores of the M1 Pro without incurring significant overhead from excessive thread management or synchronization.

## Results

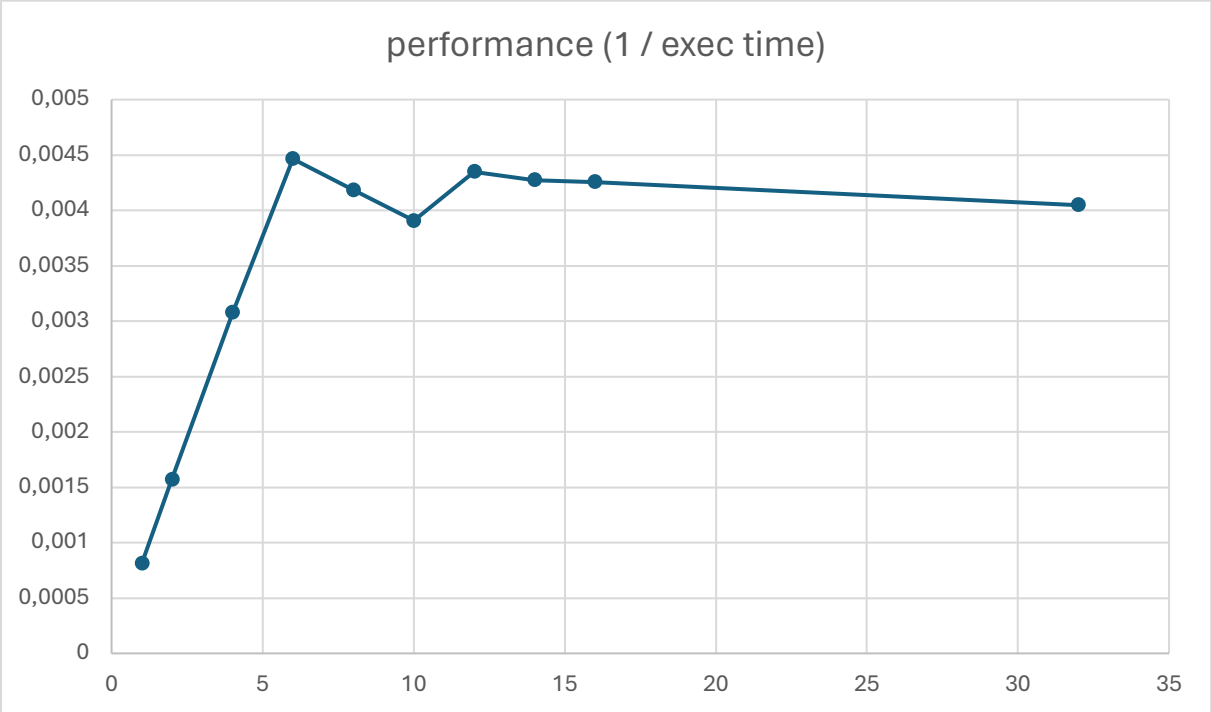
The following table summarizes the execution times (in milliseconds) as the number of threads was increased:

### Graphics:

| Threads | Execution Time |
|---|---|
| 1 | 1233 |
| 2 | 637 |
| 4 | 325 |
| 6 | 224 |
| 8 | 239 |
| 10 | 256 |
| 12 | 230 |
| 14 | 234 |
| 16 | 235 |
| 32 | 247 |

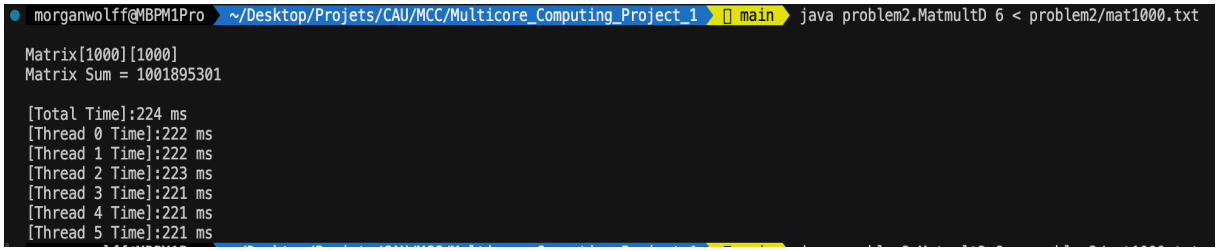


| threads | performance |
|---|---|
| 1 | 0,00081103 |
| 2 | 0,001569859 |
| 4 | 0,003076923 |
| 6 | 0,004464286 |
| 8 | 0,0041841 |
| 10 | 0,00390625 |
| 12 | 0,004347826 |
| 14 | 0,004273504 |
| 16 | 0,004255319 |
| 32 | 0,004048583 |




A graph plotting these execution times will illustrate the relationship between the number of threads and the time taken to complete the matrix multiplication.

## Execution




## Analysis

The initial reduction in execution time as the number of threads increases from 1 to 6 suggests effective parallelization, with a notable decrease in time reflecting the increased computational throughput.

Interestingly, beyond 6 threads, the execution time plateaus and slightly increases. This behavior may be attributed to several factors:

- **Overhead:** The management of a larger number of threads may introduce additional overhead that can negate the benefits of parallel execution.
- **Core Utilization:** The M1 Pro has 8 performance cores, and the optimal utilization seems to occur when the thread count is closest to this number.
- **Resource Contention:** Increasing the number of threads might lead to resource contention, particularly when the number of threads exceeds the number of physical cores.
- **Synchronization Costs:** Since the implementation involves locking, the synchronization cost can become significant when many threads attempt to access shared resources.

## Conclusion

The performance of parallel matrix multiplication on an Apple M1 Pro processor shows significant improvements up to a certain point, beyond which the overhead and contention offset the benefits of additional threads. The optimal performance was achieved with 6 threads, corresponding to the count of performance cores less than or equal to the physical core count. This insight is crucial for tuning parallel applications to leverage the hardware effectively.