2024.1 Multicore Computing, Project #1

Problem 1

Introduction

This report assesses the parallel computing performance of three distinct multithreading strategies for calculating prime numbers in Java. The computation upper limit was purposefully set at 20,000,000 to push the capabilities of the high-performance Apple M1 Pro processor. Initial testing with a smaller number space, such as 200,000, did not sufficiently challenge the system, resulting in data inadequate for deep analysis. The increased computational demand allowed for a clearer differentiation in the efficiency of the static block, static cyclic, and dynamic load balancing approaches.

Methodology

The evaluation was conducted by running Java applications implementing each load balancing strategy with a varied number of threads: 1, 2, 4, 6, 8, 10, 12, 14, 16, and 32. This provided a spectrum of data points from a single-threaded baseline up to a level of parallelism that exceeds the physical core count of the test CPU.

Each Java program tasked its threads with calculating prime numbers, utilizing their respective load balancing method to distribute the workload. Static methods pre-divided the work based on the number of threads, while the dynamic approach allowed threads to independently pull tasks from a shared pool, represented by an AtomicInteger.

System Specifications

The Java multithreading applications were evaluated on a computer with an Apple M1 Pro processor, characterized by its efficiency and robust processing capabilities. Here are the system details:

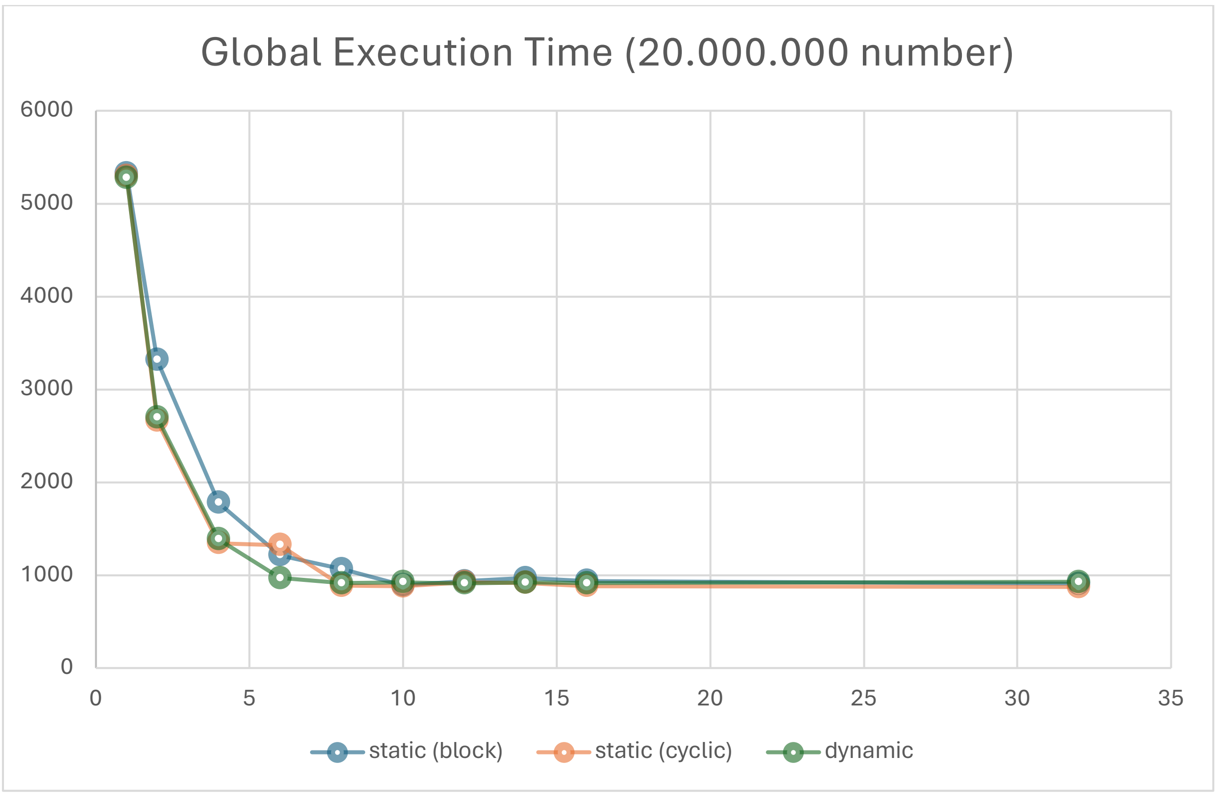
Processor: Apple M1 Pro
Clock Speed: Approximately 3.2 GHz
Core Count: 10 cores (8 performance cores, 2 efficiency cores)
RAM: 16GB of unified memory
GPU: Integrated 16-core GPU.
Neural Engine: 16-core with capabilities for advanced machine learning tasks
Memory Bandwidth: Up to 200GB/s, facilitating swift data access and manipulation.
The M1 Pro's architecture, with its combination of performance and efficiency cores, is tailored for concurrent processing and is well-suited to the demands of multithreaded operations. The integrated GPU and high-bandwidth memory further enhance the system's capacity to handle complex computational tasks efficiently.

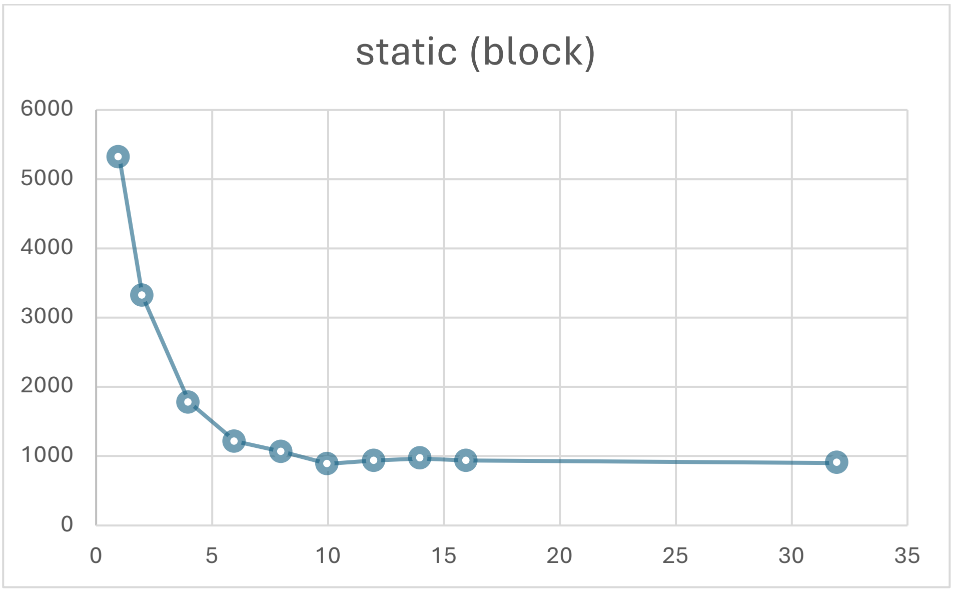
The benchmarks were conducted on macOS, which is optimized to leverage the hardware capabilities of the M1 chip, ensuring efficient thread management and optimal CPU utilization for parallel computing workloads.

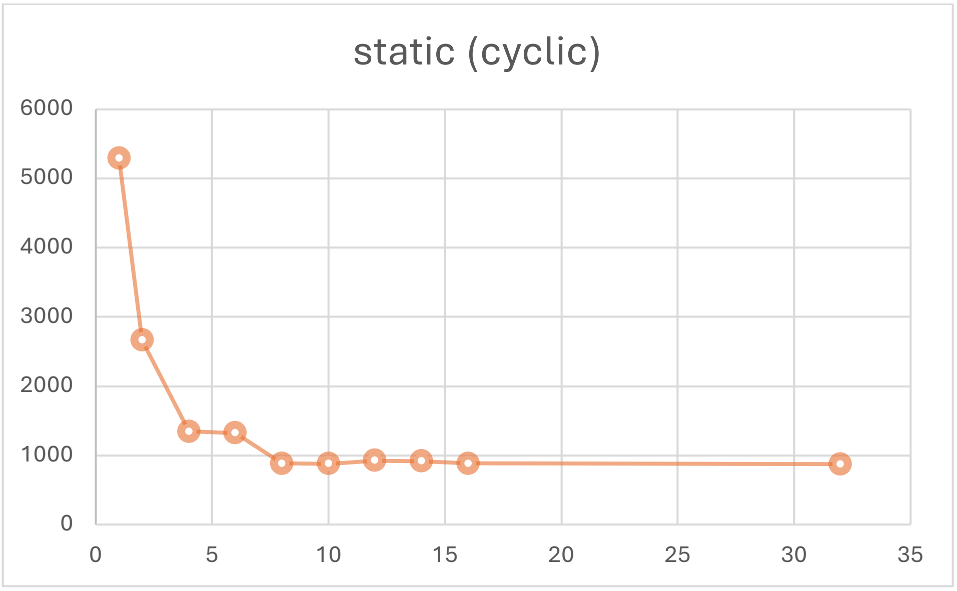
Results

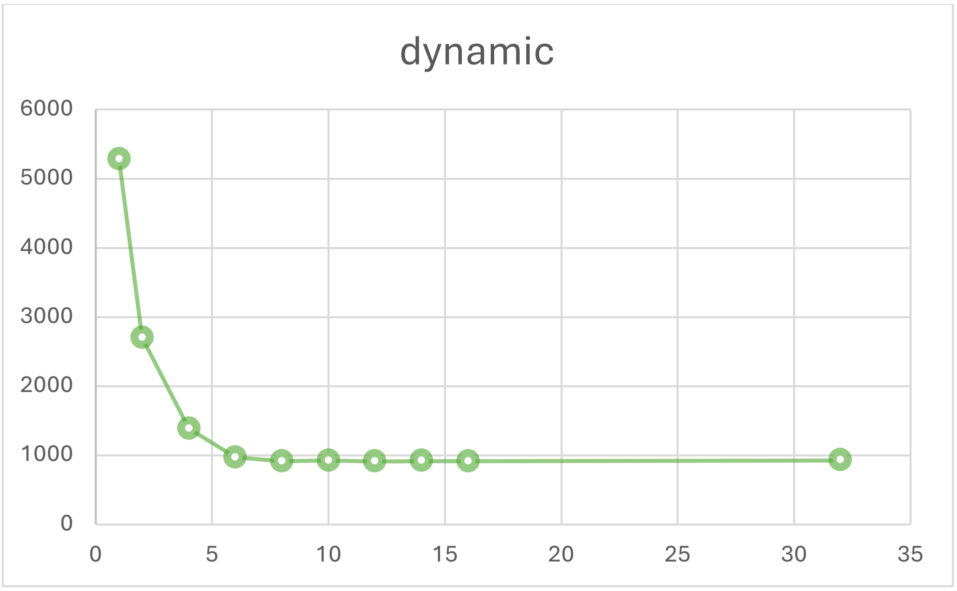
Graphics:

exec Time	static (block)	static (cyclic)	dynamic
1	5323	5296	5278
2	3318	2662	2701
4	1779	1346	1387
6	1212	1323	968
8	1066	885	913
10	891	880	923
12	935	924	911
14	970	916	919
16	936	883	914
32	906	873	928

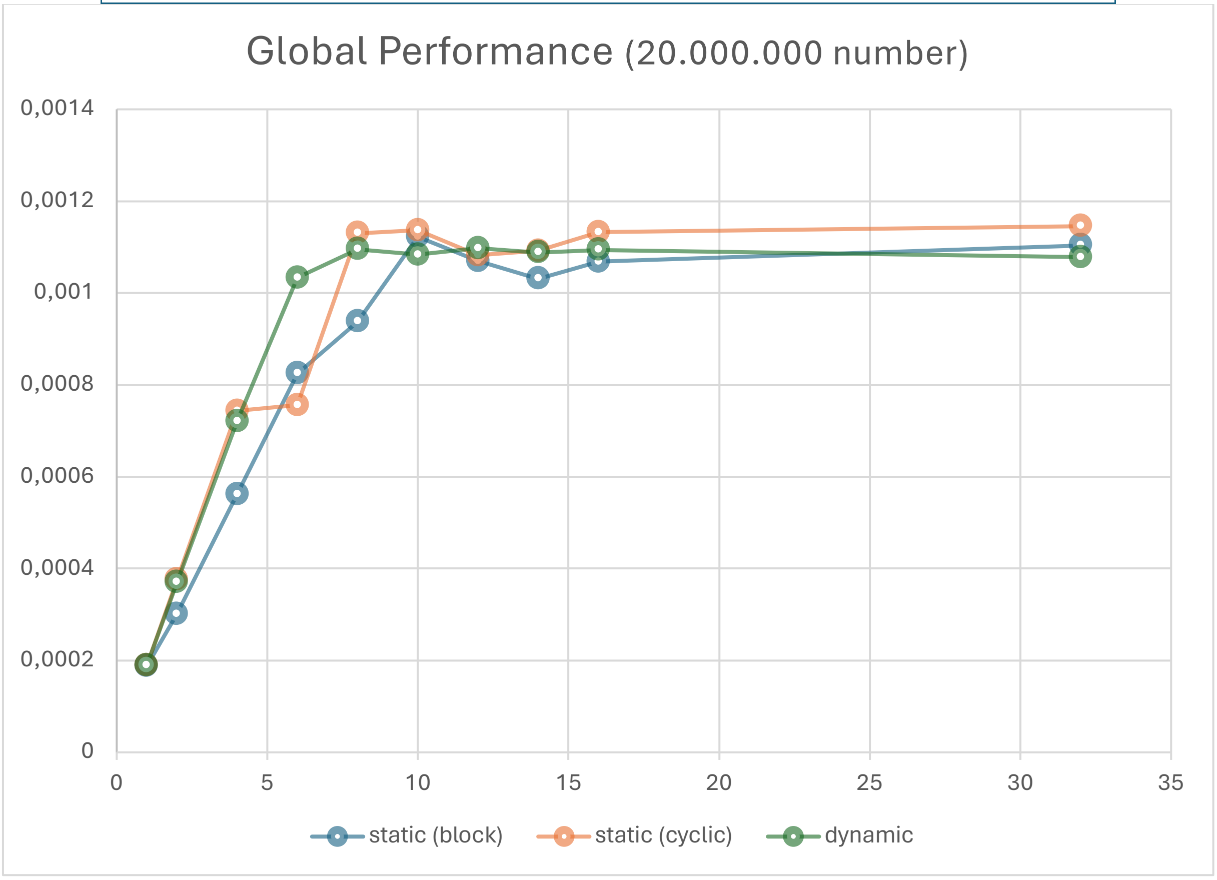


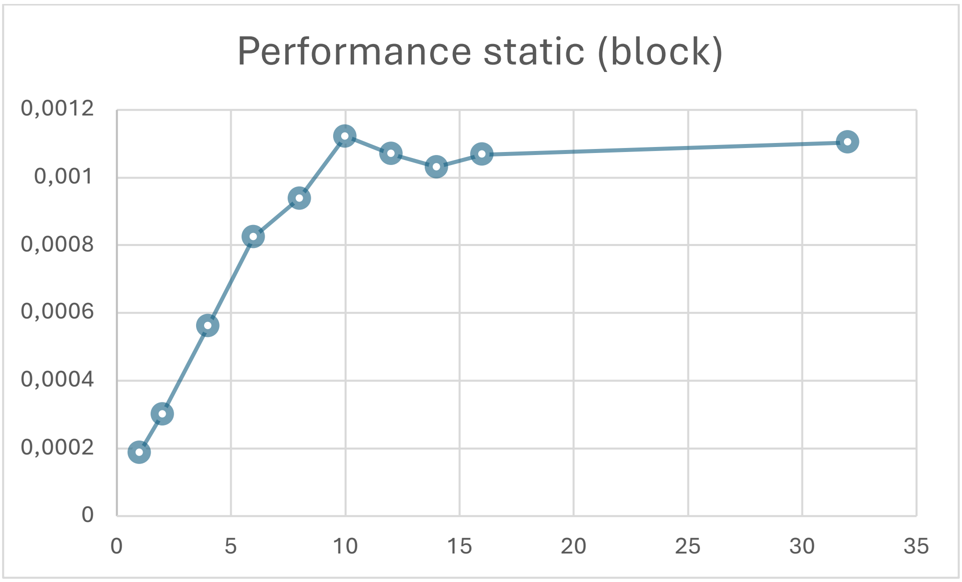

static (block)
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30
35

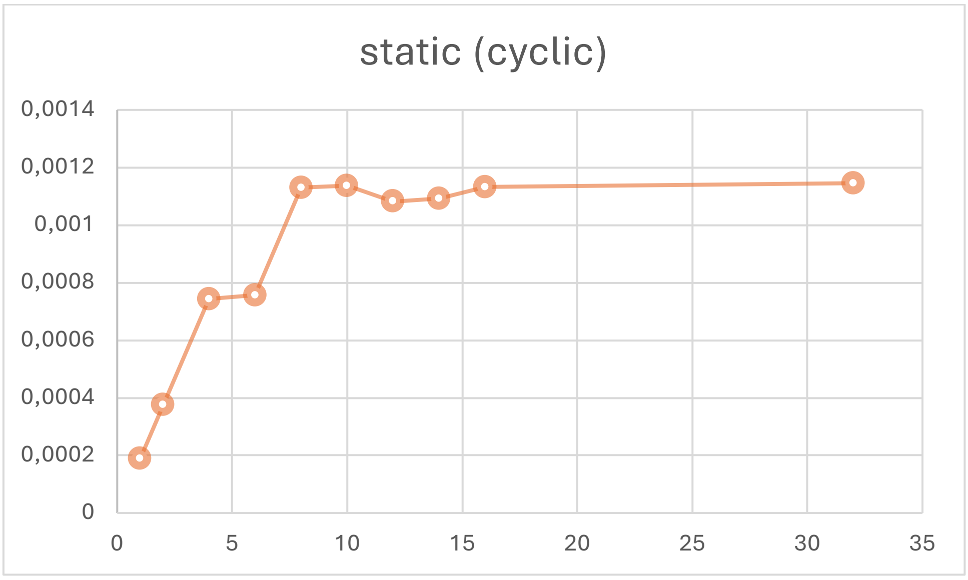

static (cyclic)
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30
35

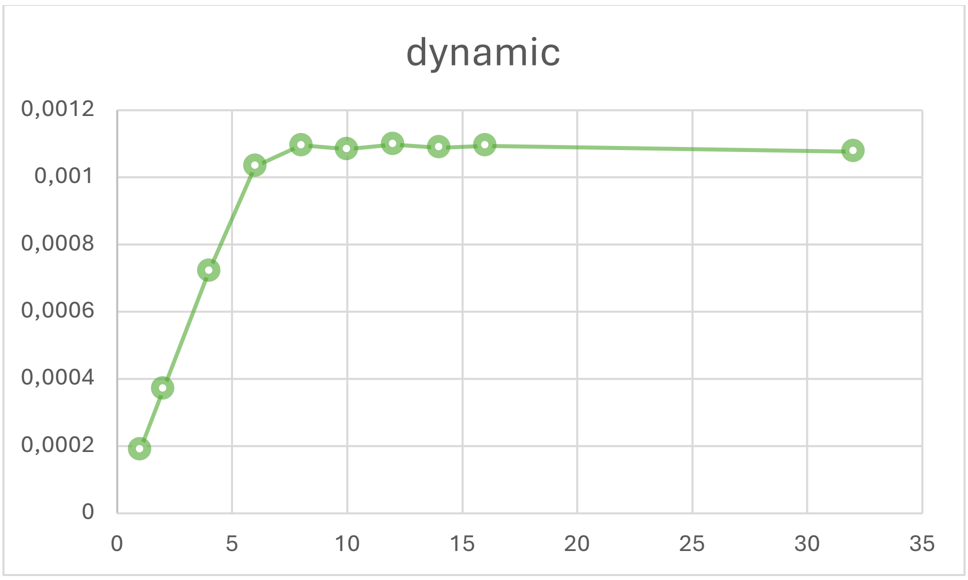

dynamic
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30
35

performance	static (block)	static (cyclic)	dynamic
1	0,000187864	0,000188822	0,000189466
2	0,000301386	0,000375657	0,000370233
4	0,000562114	0,000742942	0,000720981
6	0,000825083	0,000755858	0,001033058
8	0,000938086	0,001129944	0,00109529
10	0,001122334	0,001136364	0,001083424
12	0,001069519	0,001082251	0,001097695
14	0,001030928	0,001091703	0,001088139
16	0,001068376	0,001132503	0,001094092
32	0,001103753	0,001145475	0,001077586

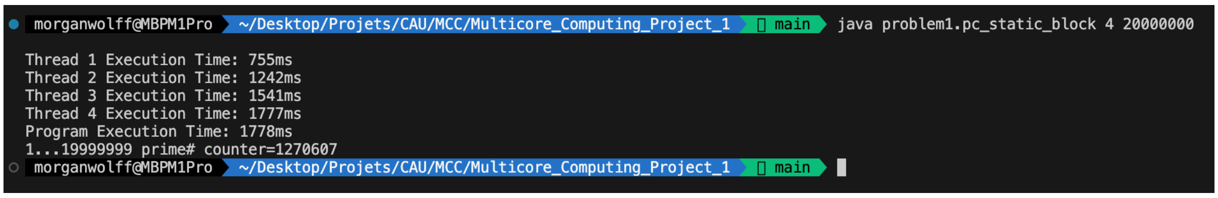



Performance static (block)
0,0012
0,001
0,0008
0,0006
0,0004
0,0002
0
0
5
10
15
20
25
30
35


static (cyclic)
0,0014
0,0012
0,001
0,0008
0,0006
0,0004
0,0002
0
0
5
10
15
20
25
30
35

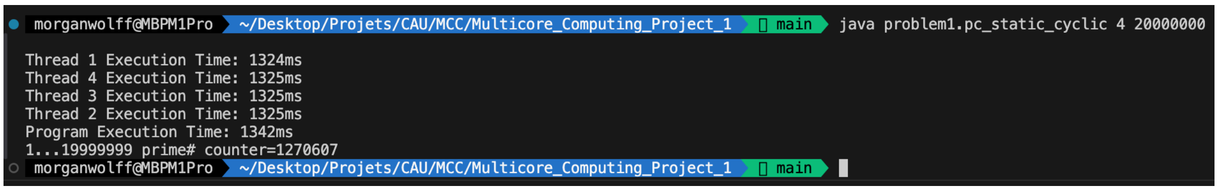

dynamic
0,0012
0,001
0,0008
0,0006
0,0004
0,0002
0
0
5
10
15
20
25
30
35

Execution



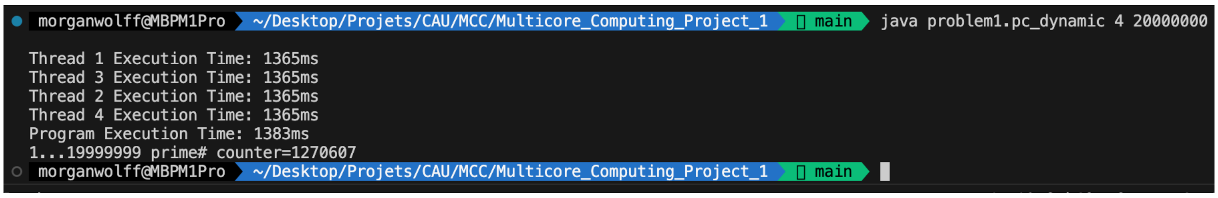
```
morganwolff@MBPM1Pro  ~/Desktop/Projets/CAU/MCC/Multicore_Computing_Project_1   main  java problem1.pc_static_block 4 20000000

Thread 1 Execution Time: 755ms
Thread 2 Execution Time: 1242ms
Thread 3 Execution Time: 1541ms
Thread 4 Execution Time: 1777ms
Program Execution Time: 1778ms
1...19999999 prime# counter=1270607
morganwolff@MBPM1Pro  ~/Desktop/Projets/CAU/MCC/Multicore_Computing_Project_1   main
```



```
morganwolff@MBPM1Pro  ~/Desktop/Projets/CAU/MCC/Multicore_Computing_Project_1   main  java problem1.pc_static_cyclic 4 20000000

Thread 1 Execution Time: 1324ms
Thread 4 Execution Time: 1325ms
Thread 3 Execution Time: 1325ms
Thread 2 Execution Time: 1325ms
Program Execution Time: 1342ms
1...19999999 prime# counter=1270607
morganwolff@MBPM1Pro  ~/Desktop/Projets/CAU/MCC/Multicore_Computing_Project_1   main
```



```
morganwolff@MBPM1Pro  ~/Desktop/Projets/CAU/MCC/Multicore_Computing_Project_1   main  java problem1.pc_dynamic 4 20000000

Thread 1 Execution Time: 1365ms
Thread 3 Execution Time: 1365ms
Thread 2 Execution Time: 1365ms
Thread 4 Execution Time: 1365ms
Program Execution Time: 1383ms
1...19999999 prime# counter=1270607
morganwolff@MBPM1Pro  ~/Desktop/Projets/CAU/MCC/Multicore_Computing_Project_1   main
```

Analyze of the results

In the analysis of the results, the findings indicate that the performance of different multithreading strategies is closely tied to the number of threads used. For both static methods, there is observed a logarithmic relationship between thread count and execution time, where times increase sharply at lower thread counts before stabilizing as more threads are added, reaching a plateau. This is explained by the system reaching the M1 Pro's concurrency limit, beyond which the overhead associated with context switching may neutralize the benefits of parallel processing.

The dynamic strategy follows a similar pattern but exhibits slight variations at higher thread counts due to overhead from managing the atomic counter. This is evident from the slight uptick in the execution time curve as thread count increases. These findings underscore the importance of striking a balance between thread count and CPU capabilities to achieve optimal performance.

Conclusion

This study delves into the importance of selecting the right multithreading strategy, considering hardware capabilities and workload demands. Through evaluating different methods, we've found that each approach has its own strengths and weaknesses.

For tasks with uniform workload distribution and heavy computation, the static cyclic decomposition method proves effective, provided minimal overhead is maintained.

On the other hand, dynamic workload distribution strategies can be advantageous for tasks with uneven workloads or when improved CPU utilization outweighs the management overhead.

Ultimately, achieving optimal performance in parallel computing requires a careful balance between thread count and hardware capabilities. Continuous experimentation and adjustments are crucial to maximize computational efficiency and resource utilization.