جامعة الأمير سلطان
PRINCE SULTAN
UNIVERSITY

College of Computer and Information Sciences (CCIS)

Term 252

CS330: Introduction to Operating Systems

Parallel Performance Evaluation Using Process-Based Execution in Linux

Instructor: **Dr. Souad Larabi-Marie-Sainte**

Team Members:

SHOUG FAWAZ ABDULLAH ALOMRAN - 223410392

LAYAN HESHAM RASHED BINDAYEL - 223410104

ARYAM YAHYA ALMANSOUR - 223410070

DANAH ABDULAZIZ ABDULMOHSEN ALTUWAIJRI - 223410151

RANA ABDULMALIK HAMAD ALFARIS - 223410567

TABLE OF CONTENTS

1. Objective

The main objective of this project is to evaluate the impact of multithreading on performance in multi-core technology. The program will be modified such that the program would calculate the cube sum of array elements along with the time taken by each thread for its operation, whether the thread is running in one core, two cores, four cores, six cores, or eight cores.

2. Introduction

Multi-core processors allow the computer to run different applications simultaneously, thus making their execution faster. Multithreading involves breaking up the program into different parts and executing them simultaneously.

The program used in this case is developed using Java language. Changes are made in this program to determine any improvement brought about by multithreading. In this code, the sum of numbers within an array is computed using different threads. As per the guidelines provided above, the squares have been calculated and added instead of adding the digits themselves. In order to test the code, this was done for one thread, two, four, six, and eight threads.

The next thing that we need to find is the execution time and speedup obtained.

3. Methodology

This phase entails modifying the initially created program code to achieve concurrency and measuring its performance. The array elements will be evenly divided into segments and assigned to individual threads.

Performance is determined by measuring execution time and speedup and percentage improvement.The program divides the input array into equal segments based on the number of threads. The computation of the cube sums by each individual thread takes place in this phase. The sum results of all the threads once completed are added together to give the final answer.

The following modifications were implemented:

- The computation was changed from summation to sum of cubes
- Array values were restricted to integers less than 100

- Support was added for multiple thread configurations (1, 2, 4, 6, and 8 threads)
- Execution time was measured for each configuration

This approach allows multiple threads to execute concurrently, improving performance when system resources are effectively utilized.

4. Experimental Setup

The experiments were conducted on the following system:

- Processor: Apple M2
- Number of cores: 8
- RAM: 8 GB
- Operating System: macOS Tahoe 26.3.1
- Java Version: OpenJDK 25 LTS (Temurin)

Each configuration was executed multiple times, and the results were averaged to ensure accuracy and consistency.

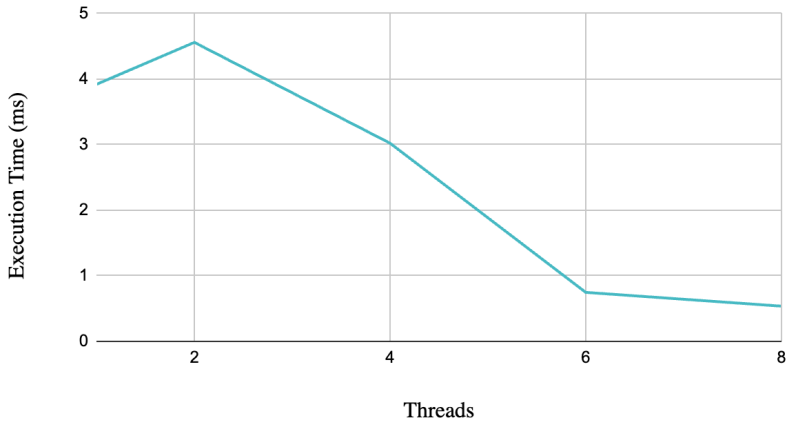
5. Results

5.1 Summary of Results

Threads	Execution Time (ms)	Speedup	% Improvement
1	3.921	1.00	0.00
2	4.560	0.86	-16.30
4	3.022	1.30	22.94
6	0.747	5.25	80.96
8	0.537	7.30	86.31

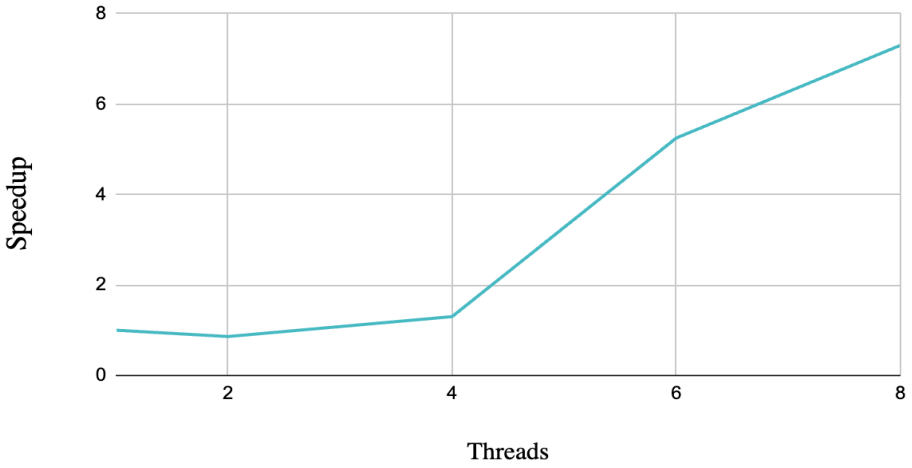
5.2 Performance Graphs

5.2.1 Execution Time vs. Number of Threads



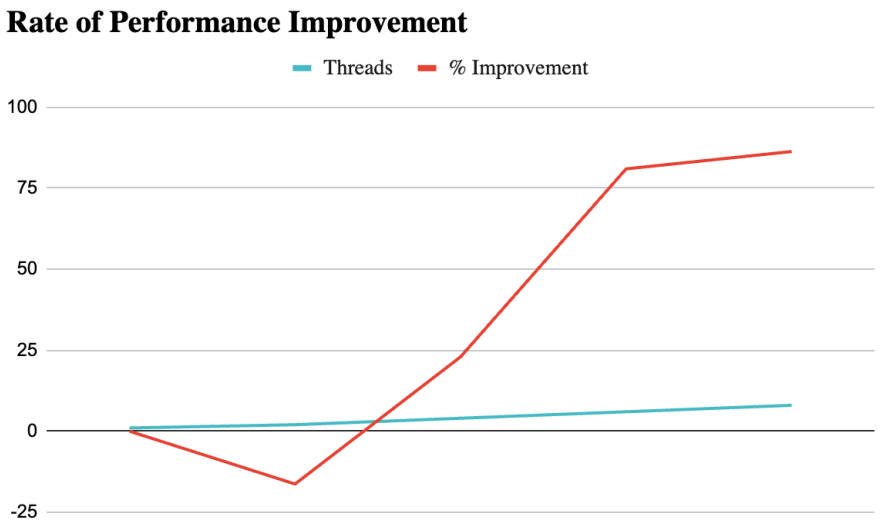
Explanation: Overall, the majority of multithreaded processes i.e. threads 4,6,8 result in reduced execution time in ms, relative to the baseline of a heavyweight single-threaded process.. Additionally, a correlation is consistently being established between reduced execution time *(ms)* and the increase in the number of the threads within a process, where a significant drop of approximately 3ms was observed as the thread number increased to 6 and 8. However, a major tradeoff can be highlighted regarding dual-threaded process models. Its execution time (ms) revealed a noticeable increase, where thread 1 resulted in an execution time of 3.921ms while thread 2 resulted in an execution time of 4.560ms. This had occurred as a result of the pure overhead caused by the creation, scheduling and synchronization of the thread that outweighed its light workload.

5.2.2 Speedup vs. Number of Threads



Explanation: From this figure, it can be highlighted that the speedup often increases as the number of threads increases. However, due to thread overhead, at 2 threads the speedup becomes slightly below 1, which means that its performance is downgraded in comparison to the single threaded case. In contrast, the speedup improves significantly with the remaining multi-threaded processes i.e 4,6,8 threads because the workload is distributed better across CPU cores.

5.2.3 Percentage Improvement vs. Number of Threads



Explanation: with 2 threads, the percentage enhancement is negative, signifying poorer performance.This is probably because of thread overhead. The higher number of threads leads to much better performance. It can be seen that the load balance for all threads improves, and the task to use the CPU resources more effectively also becomes easier. However, the most notable improvements have been noted when the number of threads used was equal to 8.

5.3 Explanation of Metrics

These results have been interpreted with regard to three important parameters, which include execution time, speedup, and percentage improvement.

1. **Execution Time (ms):** Execution time is the time required to complete the computation process. The lesser the value of execution time, the better is the efficiency of the performance.
2. **Speedup:** The term speedup refers to the measure that is used to compare the two programs' efficiency. The equation for computing the speedup factor is represented as follows:
    a. Speedup = Time taken by one thread ÷ Time taken by N threads
    b. Assuming that there are 8 threads, Speedup = 3.921 ÷ 0.537 = 7.30

As a result of the computation above, it can be seen that the application will execute seven times faster than the application executing using one thread.

3. **Percentage Improvement:** It gives us information regarding the percentage improvement in comparison with the baseline algorithm.

    a. % Improvement = $(T_1 - T\square) \div T_1 \times 100$

    where:

- T_1 = execution time with 1 thread
- $T\square$ = execution time with N threads

5.4 Interpretation of Results

The negative percentage improvement observed at 2 threads (-16.30%) indicates that performance has degraded compared to the single-threaded execution.

This occurs because:

- The overhead of creating and managing threads exceeds the benefit of parallel execution.
- Additional costs such as context switching and synchronization reduce efficiency.
- The workload required is too small to gain any benefits from parallelization in this example.

That is why there is an increase in the time of execution from 3.921 ms (1 thread) to 4.560 ms (2 threads) leading to a negative improvement ratio.

However, the benefits of a positive kind gained for 4, 6, and 8 threads prove that parallelization is more beneficial than not.

Here is how these improvements manifest themselves:

- An improvement of 22.94% for 4 threads suggests relatively good efficiency;
- An improvement of 80.96% for 6 threads means a significant boost;
- And an improvement of 86.31% for 8 threads indicates almost optimal use of the system resources.

5.4.1 Results Table

Run	Threads	Execution Time (ms)	Speedup	% Improvement
1	1	3.921	1	0
1	2	4.56	0.86	-16.3
1	4	3.022	1.3	22.94
1	6	0.747	5.25	80.96
1	8	0.537	7.3	86.31
2	1	3.671	1	0
2	2	4.207	0.87	-14.61
2	4	3.077	1.19	16.16
2	6	0.634	5.79	82.72
2	8	0.788	4.66	78.54
3	1	3.813	1	0
3	2	4.273	0.89	-12.06
3	4	3.171	1.2	16.85
3	6	0.685	5.56	82.03
3	8	0.588	6.48	84.57
4	1	3.69	1	0
4	2	4.252	0.87	-15.23
4	4	3.116	1.18	15.56
4	6	0.678	5.44	81.63
4	8	0.773	4.77	79.05
5	1	3.759	1	0
5	2	4.244	0.89	-12.89
5	4	3.139	1.2	16.49
5	6	0.592	6.35	84.26
5	8	0.611	6.15	83.74
6	1	3.702	1	0
6	2	4.217	0.88	-13.89
6	4	3.113	1.19	15.93
6	6	0.708	5.23	80.87
6	8	1.013	3.66	72.65
7	1	3.704	1	0
7	2	4.362	0.85	-17.76
7	4	3.071	1.21	17.09
7	6	0.848	4.37	77.11
7	8	0.567	6.53	84.68
8	1	3.74	1	0
8	2	4.771	0.78	-27.57
8	4	2.823	1.32	24.51
8	6	0.616	6.07	83.53
8	8	0.571	6.55	84.74
9	1	3.724	1	0
9	2	4.24	0.88	-13.84
9	4	3.446	1.08	7.49
9	6	0.631	5.9	83.05
9	8	0.65	5.73	82.56
10	1	3.924	1	0
10	2	4.205	0.93	-7.16
10	4	3.139	1.25	20.01
10	6	0.758	5.18	80.68
10	8	0.639	6.14	83.71
11	1	3.725	1	0
11	2	4.347	0.86	-16.7
11	4	3.119	1.19	16.27
11	6	0.772	4.82	79.27
11	8	0.515	7.24	86.18
12	1	3.68	1	0
12	2	4.348	0.85	-18.13
12	4	3.094	1.19	15.94
12	6	0.707	5.2	80.78
12	8	0.533	6.9	85.51

5.4.2 Sample Execution Output

```
Threads  Execution Time (ms)  Speedup    % Improvement
1        3.759                1.00       0.00
2        4.244                0.89       -12.89
4        3.139                1.20       16.49
6        0.592                6.35       84.26
8        0.611                6.15       83.74

Machine Specifications:
Processor model: Apple M2
Number of cores: 8
RAM: 8 GB
Operating system: macOS Tahoe 26.3.1
Java version: OpenJDK 25 LTS (Temurin)
```

```
Threads  Execution Time (ms)  Speedup    % Improvement
1        3.924                1.00       0.00
2        4.205                0.93       -7.16
4        3.139                1.25       20.01
6        0.758                5.18       80.68
8        0.639                6.14       83.71

Machine Specifications:
Processor model: Apple M2
Number of cores: 8
RAM: 8 GB
Operating system: macOS Tahoe 26.3.1
Java version: OpenJDK 25 LTS (Temurin)
```

```
Threads  Execution Time (ms)  Speedup    % Improvement
1        3.921                1.00       0.00
2        4.560                0.86       -16.30
4        3.022                1.30       22.94
6        0.747                5.25       80.96
8        0.537                7.30       86.31

Machine Specifications:
Processor model: Apple M2
Number of cores: 8
RAM: 8 GB
Operating system: macOS Tahoe 26.3.1
Java version: OpenJDK 25 LTS (Temurin)
```

6. Performance Analysis

Performance analysis clearly shows that multithreading greatly affects performance, but this is possible only if the number of threads is adequate for the computer's capacities.

Firstly, when using one thread, sequential execution is provided, and thus, there is maximum execution time. It should be seen as a basis of comparison.

Using two threads will affect performance negatively because managing threads entails some other tasks like creating and switching between threads. So, the benefits associated with using multiple threads will not be achieved.

With regards to four threads, we observe improved performance because of equal distribution of tasks among all the threads. Thus, it provides good performance because of effective usage of CPU resources.

A noticeable improvement is provided by the use of six threads since the load becomes well balanced and multithreading becomes efficient. Thus, its advantages exceed the disadvantages.

In conclusion, it would be appropriate to state that the highest performance could be delivered by eight threads. This is because there are eight cores on the computer.

7. Maximum Effective Thread Count

Performance was optimal when the eight threads were used, a number that equates to the number of CPU cores available.

From this experiment, one can learn that optimal performance is attained by using threads whose number equals the number of cores. More threads will reduce the efficiency since there will be more scheduling and overhead costs incurred in such an attempt.

8. Conclusion

From this project, one can realize how important multithreading is to improve the efficiency of applications when dealing with multi-core processors. Multithreading helps to minimize the computation time for the tasks performed.However, the findings of the test also indicate that multithreading should be implemented in a strategic manner. An abundance of threads may also cause overhead, reducing efficiency levels.

In the end, it becomes evident from the test that the highest efficiency is achieved when the number of threads used equals the number of CPU cores present in the machine.

References

Alomran, S. F., Bin Dayel, L., Almansour, A. Y., & Altuwajiri, D. (2026). *CS330 parallel performance evaluation in Linux project* [Source code]. GitHub. https://github.com/Shoug-Alomran/CS330-Parallel-Performance-Evaluation-in-Linux-Project

Alomran, S. F. (2026). *Operating systems project documentation*. https://operating-systems.shoug-tech.com/