Faculty of engineering and technology

Electrical and Computer Engineering Department

OPERATING SYSTEMS - **ENCS3390**

TASK - 1

Instructor: Dr.**Bashar Tahayna**
Section: 3

Name: Husain Abugosh
ID: 1210338

RESULT OF EACH APPROACH:

Approach	First	Second	Third	Fourth	Fifth	Average	Throughput
Naive approach	0.005964 sec	0.00595 sec	0.005894 sec	0.005163 sec	0.00336 sec	**0.0052662 sec**	**189.89**
----------	----------	----------	----------	----------	----------	----------	----------
multiple child processes (num_process = 2)	0.004682 sec	0.004677 sec	0.00424 sec	0.004688 sec	0.003197 sec	**0.0042968 sec**	**232.73**
multiple child processes (num_process = 4)	0.003526 sec	0.003624 sec	0.003531 sec	0.004157 sec	0.003859 sec	**0.0037394 sec**	**267.42**
----------	----------	----------	----------	----------	----------	----------	----------
multiple joinable threads (num_threads = 2)	0.003154 sec	0.003156 sec	0.003529 sec	0.001913 sec	0.003555 sec	**0.0030614 sec**	**326.64**
multiple joinable threads (num_threads = 4)	0.001872 sec	0.002985 sec	0.003512 sec	0.001840 sec	0.002118 sec	**0.0024654 sec**	**405.61**
----------	----------	----------	----------	----------	----------	----------	----------
multiple detached threads (num_threads = 2)	0.000175 sec	0.000069 sec	0.000159 sec	0.000179 sec	0.000087 sec	**0.000134 sec**	**7462.68**
multiple detached threads (num_threads = 4)	0.003175 sec	0.000228 sec	0.00036 sec	0.000848 sec	0.000116 sec	**0.0009454 sec**	**1057.75**

***Before I start, I want to notice that I did not use Linux VM cause I have a macOS (which is based on a Unix-like operating system).**

Introduction:

This report analyzes the performance of matrix multiplication using various parallelization approaches. The execution times for different approaches and thread/process counts were measured and compared. The goal is to understand the impact of concurrency on execution time and determine the best approach for different matrix widths.

Variation in Execution Time for the Same Approach:

we can notice that there is a Variation Execution Time for the Same Approach and **even** for the **samme** number of process and threads an this could be :

- **Resource Contention:** The execution times for the same approach and the same number of processes or threads exhibit variations. This can be attributed to resource contention, where multiple concurrent entities compete for CPU time and other resources.

- **Scheduling and Overheads:** Operating system scheduling, context switching, and the creation/termination of threads/processes introduce overheads. These overheads vary across runs, affecting execution times.

- **System Load:** System load from other processes can impact performance. Higher system load during a test may result in slower execution times.

- **Memory Access Patterns:** The way memory is accessed can affect execution speed. For example, if matrix data is spread out in memory, it might take longer to access due to page faults or cache misses.

- **CPU Caching**: Modern CPUs use caches to store data that are likely to be used soon. If the data used by your program is already in the cache (because of a previous run or other activity on the system), the program may run faster.

Comparison of Same Approach with Different Numbers of Processes or Threads:

- **Diminishing Returns:** increasing the number of processes or threads may lead to better CPU utilization up to a point. Beyond that, the overhead of managing concurrent entities can outweigh the benefits, especially when the number of entities exceeds the number of CPU cores.

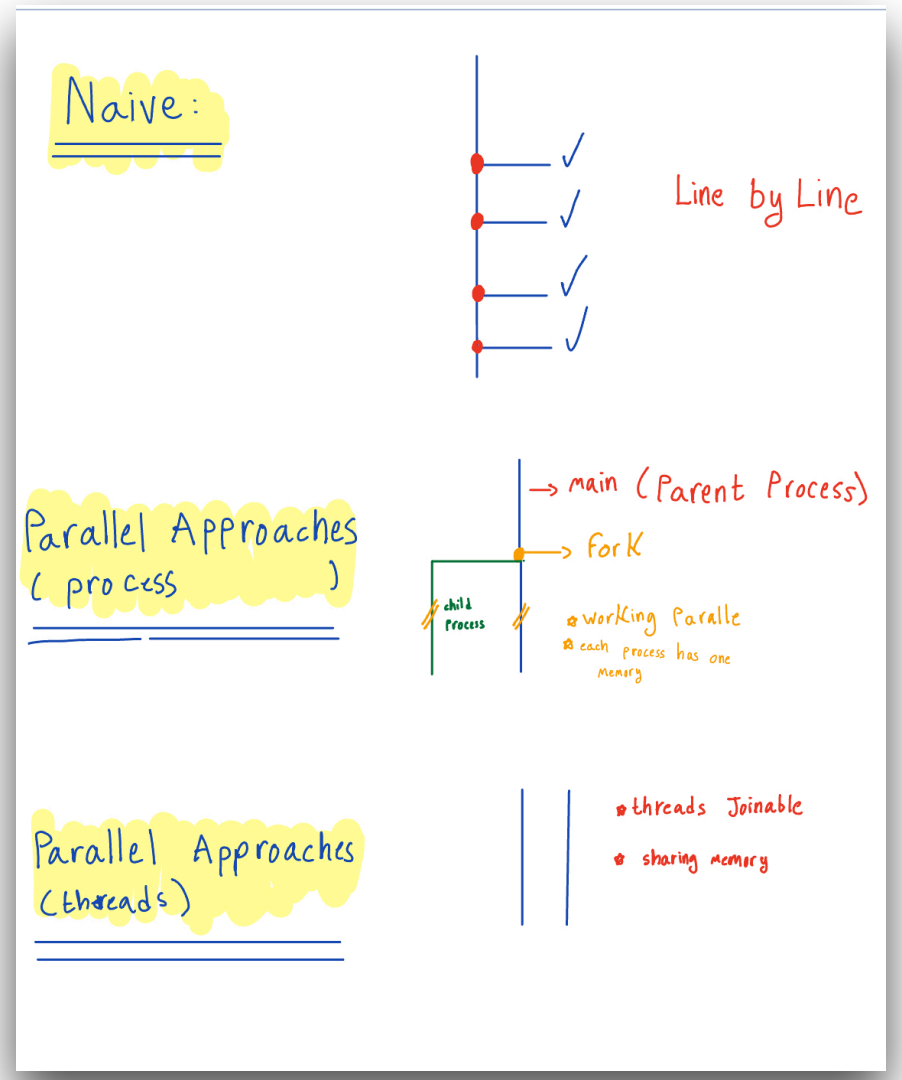
- **Concurrency vs. Parallelism:** With more threads than cores, concurrency (context-switching) may occur instead of true parallelism, impacting speed-up.

Comparison of Approaches with Each Other

- **Naive vs. Parallel Approaches:** The naive (sequential) approach typically takes longer as it doesn't utilize multiple CPU cores. Parallel approaches distribute the work among multiple cores, reducing execution time.

- **Processes vs. Threads:** Processes have higher overhead than threads, as they require more memory and have greater start-up and communication costs. Threads share the same memory space and are lighter, resulting in potentially faster execution times.

- **Detached Threads:** Detached threads can be faster as they can be terminated without needing the main thread to join them, reducing overhead. However, synchronization can be more challenging.



Best Approach for Each Matrix Width:

- **Small Matrices:** Overhead may dominate execution time, so fewer threads or processes may be optimal for small matrices.

- **Large Matrices:** Parallelization benefits are more pronounced with larger matrices. However, the number of threads/processes should not exceed the available CPU cores to avoid inefficiency.

Conclusion:

Matrix multiplication performance can vary significantly based on the chosen parallelization approach and the number of threads/processes used. Understanding these variations and selecting the appropriate approach for different scenarios is crucial for optimizing performance.

The relationship between parallelization and execution speed is complex and depends on various factors, including CPU resources, task dependencies, algorithm design, and data size. Amdahl's Law reminds us that the speedup from parallelization is limited by the fraction of the code that cannot be parallelized. Therefore, careful consideration of these factors is essential.

This report has provided valuable insights into the factors affecting execution time, including the analysis of different parallelization approaches and the impact of varying the number of threads/processes. The performance measurements and comparisons have shed light on the advantages and trade-offs associated with each approach.

To further optimize matrix multiplication performance, it is recommended to conduct additional testing with larger data sets and different hardware configurations. Additionally, exploring advanced parallelization techniques and load balancing strategies may lead to enhanced speedup in specific scenarios.

In conclusion, matrix multiplication is a computationally intensive task, and its performance can be significantly improved through parallelization. However, the effectiveness of parallelization depends on several factors, and a thoughtful approach is required to achieve optimal results.

Resources:

- **https://www.youtube.com/playlist?list=PLfqABt5AS4FmuQf70psXrsMLEDQXNkLq2**
- **https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm**
- **Chat GPT**