APROP REPORT

Henrique (1200883) and Alexander (1222703)

The purpose of this work is to benchmark three methods of matrix updates -sequential, task-based, and block-based - by analysing their execution times under different configurations of block sizes (BS), matrix sizes (N), and number of threads (NUM_THREADS). The goal is to understand the performance impact of various parallelisation strategies in OpenMP, achieved by varying each parameter independently. The results are written to a CSV file for further analysis, and an additional Python script generates 3D graphs for a clearer visualization of the performance trends. The insights gained help assess how the choice of block sizes, matrix dimensions, and thread counts influence the computational efficiency of matrix operations.

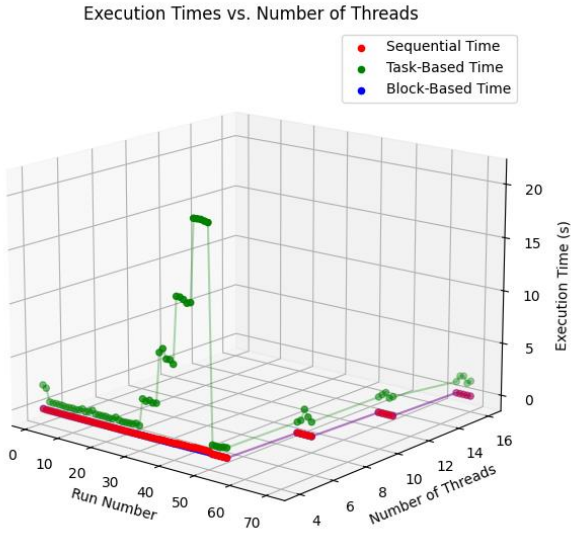
The benchmarks were conducted on Henrique's and Alex's machines, using GCC and LLVM compilers.

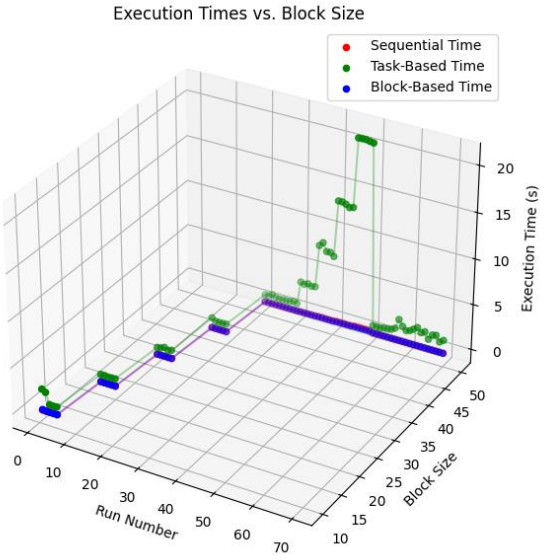
Below are the key observations from the results using GCC:

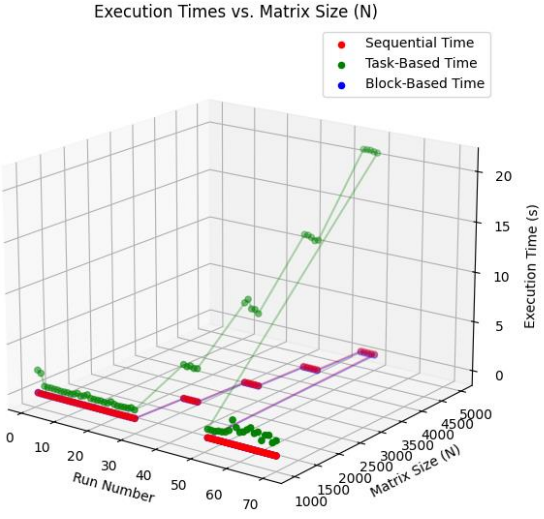
- Sequential Processing:
  - For small matrix sizes (N = 1000), the sequential method shows the shortest execution times. However, as the matrix size increases, execution times rise proportionally, highlighting the scalability limitations of this approach. With larger matrices (N = 5000), the sequential method becomes significantly slower, although it remains within the performance capacity of each machine.
- Task-Based Parallelism:
  - Task-based parallelism is less efficient than sequential processing for smaller matrices like N = 1000 due to the overhead from task management, thread synchronization, and context switching. However, as the matrix size increases (N = 2000 and above), task-based parallelism becomes more effective, demonstrating improved performance relative to the sequential approach. Despite this, the gains are not linear, and the method exhibits high execution times with larger matrices (N = 5000), indicating significant overhead.
- Block-Based Parallelism:
  - The block-based approach yields promising results, particularly when the block size (BS) is well chosen. For smaller matrices (N = 1000), block-based methods outperform task-based methods in terms of execution time. The approach remains relatively efficient even as matrix sizes grow, although its success depends on selecting an optimal block size. For instance, with larger matrices and a BS of 50, block-based methods demonstrate better-balanced execution times. Overall, block-based parallelism is more efficient than task-based methods for larger matrices, making it a preferred strategy for handling extensive data sizes.

In the next few pages, it will be presented the graphs produced from the results_gcc.csv of each person's machine (Henrique and Alex) for the GCC compiler:

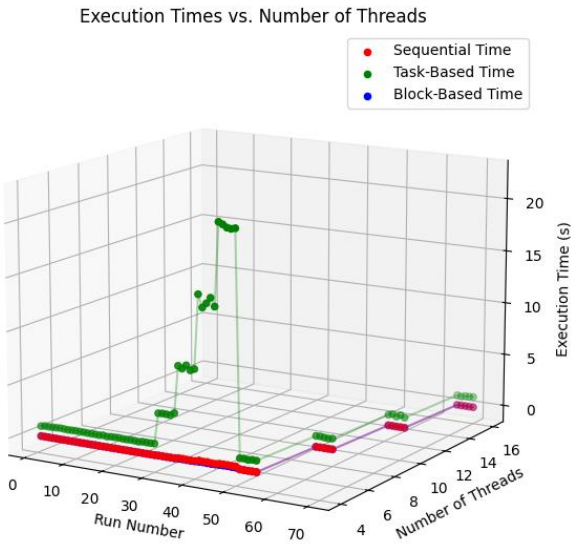
- Henrique’s performance

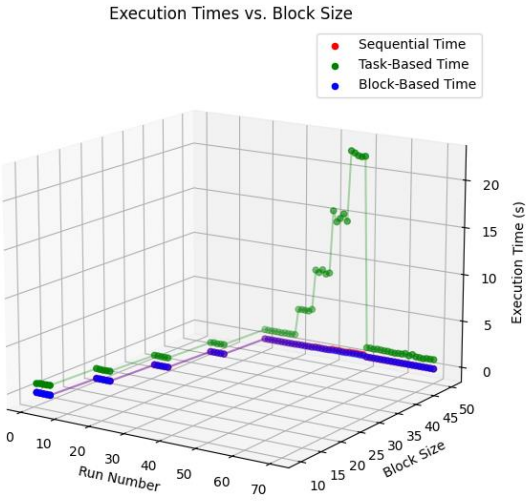


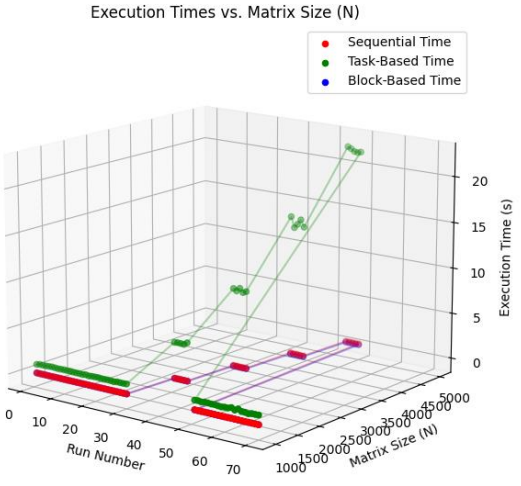




- Alex’s performance:







Below are the key observations from the results using LLVM Clang compiler:

When testing the LLVM-compiled version of our code, we encountered significant performance issues, particularly with task-based parallelism. Despite using the same code as for the GCC tests, the LLVM compiler yielded much slower execution times for task-based operations compared to both sequential and block-based methods.

Additionally, attempts to increase the problem size by raising the value of N from 1000 to 2000 (run 30), similar to the GCC experiments, encountered further complications. Each time N exceeded 1000, the terminal output displayed "Killed," indicating that the process was forcefully terminated. This issue prevented further evaluation of the LLVM-compiled code with larger values of N, thus restricting our ability to make a direct comparison with the GCC results. The "Killed" may suggest resource constraints or memory issues as the problem size increases in the LLVM-compiled version, which we were unable to resolve within the scope of our tests. This limitation highlighted a significant discrepancy between the two compilers in terms of handling larger computational loads.

```
make: *** [makefile:43: ex5_llvm_run] Killed
```

Although these difficulties happened, below is the analysis performance for the 30 runs we could do using the LLVM Clang compiler:

- Sequential Method Analysis:
    - The sequential method exhibits the fastest execution times across all runs in both results, generally going between 0.007 to 0.025 seconds.
    - This method does not benefit from parallelisation and is limited by the processing power of a single thread. However, it also has the lowest overhead compared to the other methods, making it efficient for small problem sizes (like N=1000N=1000).
    - While efficient, the sequential approach will likely become a bottleneck as N increases due to the lack of parallel processing.

- Task-Based Method Analysis:
    - Task-based parallelisation shows higher execution times compared to both sequential and block-based methods, with a range of approximately 2.1 to 4.8 seconds in Henrique’s results and 2.5 to 3.7 seconds in Alex's results.
    - This method breaks the workload into smaller tasks, which can lead to significant overhead if the number of tasks is high and the workload per task is relatively low.
    - The task-based approach is less efficient than the block-based method for this problem size due to overhead from task management and synchronization. It might perform better when the workload per task is

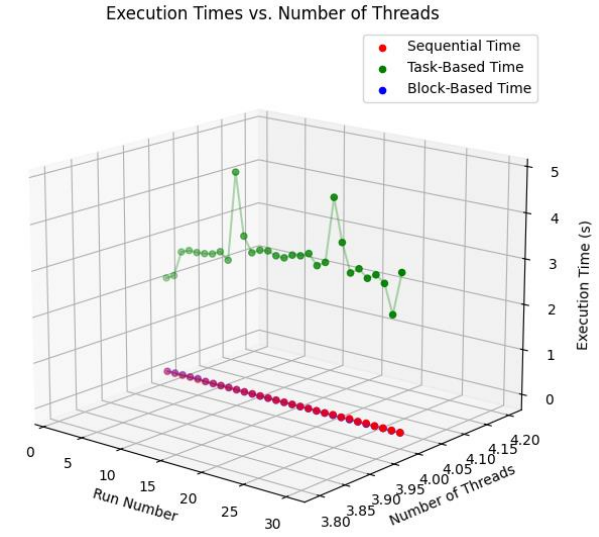
more balanced or larger, but the resource constraints seem to limit its effectiveness.

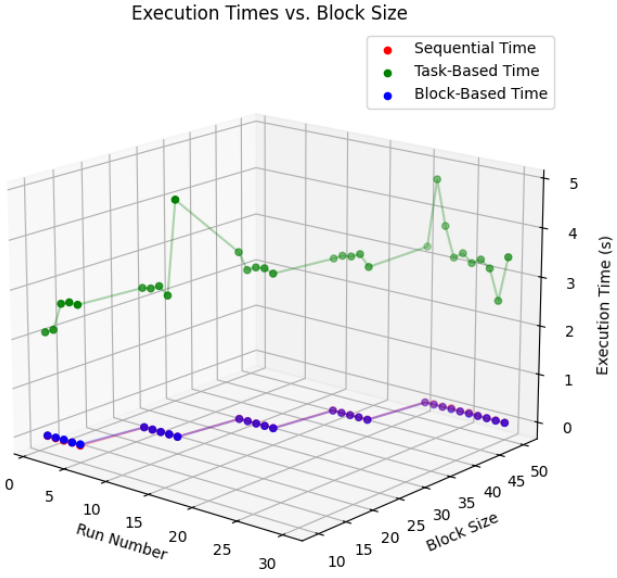
- Block-Based Method Analysis:
  - Block-based parallelization generally performs better than the task-based method, with execution times between 0.0029 and 0.037 seconds.
  - It divides the workload into blocks that are processed in parallel, offering a more balanced trade-off between parallelism and overhead.
  - For lower block sizes (e.g., BS=10BS=10), block-based times are higher due to increased management overhead. As Block Size increases (e.g., to 50), the performance improves since there is less overhead from frequent synchronization, and each thread has more work.
  - The block-based approach is more efficient for the given problem size, providing better performance as long as the blocks are well-sized relative to the available resources.

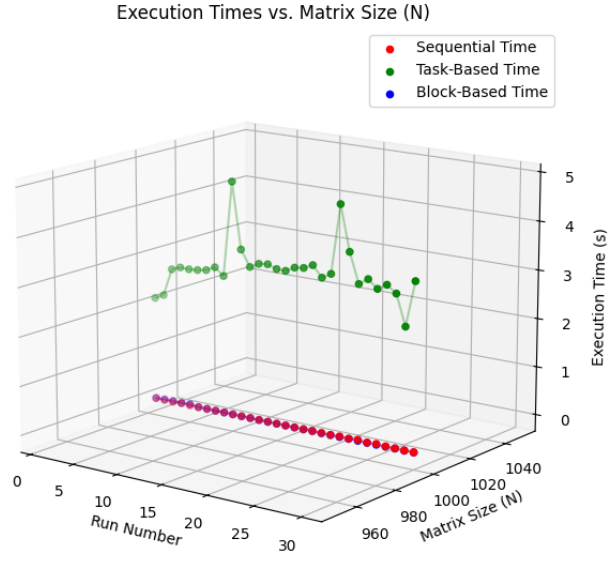
- The sequential method remains the fastest due to its simplicity, but it lacks scalability.
- The task-based method incurs higher overhead due to task management, especially noticeable for smaller workloads.
- The block-based method balances parallelisation and overhead better than the task-based method, providing faster execution times in many cases.

In the next few pages, it will be presented the graphs produced from the results_gcc.csv of each person's machine (Henrique and Alex) for the LLVM Clang compiler:

- Henrique’s performance







- Alex’s performance: