|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Versions** | **Grid Size** | **Fish** | **Sharks** | **Steps** | **Threads** | **Shark**  **Breed Time** | **Fish**  **Breed Time** | **Starve Time** | **Duration** |
| Normal | 50x50 | 50 | 50 | 50 | None | 4 | 3 | 4 | 6.711s |
| SpeedUp | 50x50 | 50 | 50 | 50 | 1 | 3 | 3 | 4 | 6.337s |
| Normal | 50x50 | 50 | 50 | 50 | None | 4 | 3 | 4 | 7.175s |
| SpeedUp | 50x50 | 50 | 50 | 50 | 2 | 3 | 3 | 4 | 6.668s |
| Normal | 50x50 | 50 | 50 | 50 | None | 4 | 3 | 4 | 6.335s |
| SpeedUp | 50x50 | 50 | 50 | 50 | 4 | 3 | 3 | 4 | 6.336s |
| Normal | 100x100 | 100 | 100 | 50 | None | 4 | 3 | 4 | 23.202s |
| SpeedUp | 100x100 | 100 | 100 | 50 | 8 | 3 | 3 | 4 | 22.931s |
| Normal | 100x100 | 100 | 100 | 50 | None | 4 | 3 | 4 | 22.930s |
| SpeedUp | 100x100 | 100 | 100 | 50 | 10 | 3 | 3 | 4 | 22.732s |

# **TIMED RESULTS FOR WA-TOR PROJECT THREADED AND NON-THREADED**

## Description of Measurements

**The performance of the Wa-Tor simulation was measured using:**

* Grid Sizes: 50x50 and 100x100.
* Entity Counts: Equal numbers of fish and sharks (e.g., 50 fish and 50 sharks for 50x50 grid).
* Simulation Steps: 50 steps for consistency.
* Threads: The threaded version was run with 1, 2, 4, 8, and 10 threads for parallel processing.
* Timing: Execution time (in seconds) was recorded using the time package in Go.

## Explanation of Results

**No Speedup for Low Threads:**

* With 1 or 2 threads, the speedup was marginal or barely noticeable (e.g., ~6.7s vs. ~6.3s). This is due to the overhead of thread creation and synchronisation, which can outweigh the benefits of parallelism at smaller workloads.

**4 Threads Show No Additional Gain:**

* The grid size and workload are not large enough to fully benefit from parallelisation, as each thread processes fewer rows.

**Slight Improvement with 8-10 Threads:**

At higher grid sizes, the workload becomes sufficiently larger for threading to show slight speedup (e.g., ~22.9s vs. ~22.7s with 8-10 threads). However, the improvement is marginal because:

* The grid still requires frequent synchronisation to merge thread results.
* Core contention occurs as threads compete for limited CPU resources (e.g., on a dual-core Intel i3).

## Why Speedup is Limited

**Concurrency Overheads:**

Creating threads, dividing the grid, and synchronising updates introduce significant overhead, especially for smaller grid sizes.

**Core Limitation:**

My Intel core i3 processor has only 2 physical cores 4 logical limiting the maximum concurrent work that can be performed. Using more threads than cores leads to context-switching overhead, reducing potential gains.

**Uneven Work Distribution:**

Threads process sections of the grid, and some sections may have fewer entities, leading to idle time for some threads while others handle more work.

**Synchronised Grid Updates:**

After processing fish and sharks, the grid state must be synchronised across threads. This reduces the benefits of parallel processing.

## Conclusions

* 50x50 Grid: The threaded version offers no meaningful advantage due to high overhead relative to the workload.
* 100x100 Grid: Small speedups (~1-2%) are observed with 8-10 threads, but the gains are marginal due to core limitations and synchronisation overhead.
* Concurrency is effective only for larger workloads and when the number of threads aligns with the available CPU cores.
* Overheads from synchronisation and thread management can outweigh benefits for smaller grid sizes or fewer entities.
* Optimal thread count depends on workload size and hardware capabilities, typically matching or slightly exceeding the number of available CPU cores.

## References

1. GeeksforGeeks. (2019). *Difference between Concurrency and Parallelism*. [online] Available at: <https://www.geeksforgeeks.org/difference-between-concurrency-and-parallelism/>.
2. freeCodeCamp.org. (2022). *Concurrent Programming in Go – Goroutines, Channels, and More Explained with Examples*. [online] Available at: <https://www.freecodecamp.org/news/concurrent-programming-in-go/>.
3. eamonngaynor (2018). *GitHub - eamonngaynor/CDDProject: Wator simulation written in C++ and optimized using OpenMP*. [online] GitHub. Available at: <https://github.com/eamonngaynor/CDDProject>.
4. GeeksforGeeks. (2019). *Goroutines - Concurrency in Golang*. [online] Available at: <https://www.geeksforgeeks.org/goroutines-concurrency-in-golang/>.
5. Wikipedia Contributors (2023). *Wa-Tor*. Wikipedia.

‌