




Wa-Tor Speed-up Report



**SE
TU**

Ollscoil
Teicneolaíochta
an Oirdheiscirt

South East
Technological
University



Student Name: Daniel Giedraitis
Student Number: C00260331
Academic Year: 2023/2024
Lecturer: Joseph Kehoe
Subject: Concurrent Development

Summary

This report details the implementation of the "Wa-Tor" simulation on a Linux platform using the gcc compiler in accordance with the project requirements. The project aimed to simulate ecological interactions between sharks and fish on the toroidal planet of Wa-Tor, following the principles elucidated by A.K. Dewdney in the Scientific American article "Computer Recreations.", was executed using OpenMP for parallelization. The benchmark utilized a Windows Subsystem for Linux (WSL) operating on an AMD Ryzen 7 5700U processor, equipped with 8 cores, ensuring a robust computational foundation for the simulation.

Parameters of the System Used

The benchmark employed a Windows Subsystem for Linux on a Windows machine, operating Ubuntu 22.04.3 LTS. The system encompasses an AMD Ryzen 7 5700U processor with 8 cores, executing the simulation within this environment.

Global Variables in the Project Compilation

Constants:

- **maxDuration:** Maximum duration of simulation steps.
- **maxThreads:** Maximum number of threads used.
- **fishBreed, sharkBreed:** Time units for fish and shark reproduction.
- **energyGain:** Energy acquired by sharks when feeding.
- **starve:** Time period before a shark dies due to starvation.
- **xdim, ydim:** World dimensions.
- **WindowXSize, WindowYSize:** Window size for graphical display.
- **tileXSize, tileYSize:** Tile sizes for visualization.

Structures and Variables:

TileType: Enumeration defining tile types (Ocean, Fish, Shark)

Tile: Structure representing a tile in the simulation

window: Render window for visualization

display[][]: Array storing graphical representation

tiles[][]: Array defining the world as tiles

DecisionData[]: Array holding decision data for threads

Methodology (Benchmarking & Parallelization)

The simulation utilized C++ with OpenMP for parallelization. OpenMP directives facilitated parallel execution, allowing the simulation to run concurrently across multiple threads. The methodology incorporated:

Thread-based Execution: Utilized OpenMP for parallelizing the movement of creatures (fish and sharks) within the simulation.

Benchmarking: Measured and recorded durations for various thread counts to evaluate performance.

Results

The simulation was ran 100 times for each thread count (1, 2, 4, 8).

Durations (ms):

1 Thread: [55, 55, 59, 63, 70, 85, 75, 78, 79, 79, 81, 81, 80, 79, 78, 80, 80, 79, 81, 79, 75, 73, 70, 69, 74, 71, 67, 61, 59, 59, 61, 61, 60, 59, 60, 61, 64, 65, 65, 66, 66, 68, 72, 72, 71, 69, 70, 69, 73, 71, 71, 68, 68, 67, 72, 70, 69, 66, 66, 65, 69, 67, 67, 64, 64, 64, 68, 68, 66, 65, 64, 66, 69, 68, 68, 66, 66, 66, 70, 69, 68, 66, 66, 67, 71, 69, 68, 68, 68, 66, 70, 69, 68, 65, 65, 64, 68, 67, 66, 64]

2 Threads: [41, 39, 45, 49, 52, 47, 61, 50, 51, 65, 67, 51, 66, 51, 64, 63, 63, 64, 53, 52, 59, 56, 54, 46, 48, 45, 45, 41, 43, 41, 42, 44, 44, 44, 43, 43, 48, 44, 48, 50, 45, 45, 55, 47, 55, 54, 46, 54, 56, 46, 53, 52, 52, 43, 55, 54, 52, 50, 48, 49, 45, 51, 50, 50, 48, 45, 51, 43, 54, 49, 43, 51, 53, 51, 51, 45, 50, 43, 47, 47, 51, 50, 50, 47, 46, 47, 52, 51, 43, 49, 54, 51, 45, 44, 45, 48, 52, 50, 43, 44]

4 Threads: [34, 33, 33, 36, 39, 41, 43, 45, 37, 46, 45, 47, 45, 43, 46, 46, 47, 46, 48, 43, 42, 40, 37, 38, 59, 40, 38, 34, 35, 33, 35, 34, 34, 34, 33, 35, 35, 36, 37, 36, 37, 37, 39, 39, 40, 34, 39, 34, 41, 40, 40, 39, 45, 42, 42, 39, 37, 36, 36, 36, 39, 39, 38, 37, 35, 34, 37, 38, 36, 35, 36, 38, 39, 37, 39, 35, 37, 39, 39, 39, 38, 38, 36, 36, 40, 38, 37, 35, 40, 34, 39, 38, 39, 37, 39, 36, 35, 37, 38, 44]

8 Threads: [33, 34, 34, 37, 37, 35, 38, 35, 38, 40, 34, 39, 36, 37, 37, 37, 40, 39, 39, 38, 37, 35, 36, 35, 37, 35, 36, 35, 34, 34, 34, 34, 34, 34, 34, 35, 34, 34, 34, 34, 35, 35, 35, 36, 35, 35, 34, 36, 36, 35, 34, 38, 35, 40, 45, 35, 36, 38, 33, 35, 35, 35, 36, 35, 34, 35, 36, 35, 37, 34, 36, 37, 34, 37, 36, 36, 35, 39, 36, 34, 38, 35, 33, 35, 35, 35, 34, 37, 34, 36, 35, 36, 34, 35, 35, 35, 35, 35, 35]

Table showing min, max and average duration of each thread.

	Threads			
	1	2	4	8
Minimum (ms)	55	39	33	33
Maximum (ms)	85	67	59	45
Average (ms)	68.56	49.62	38.6	35.66

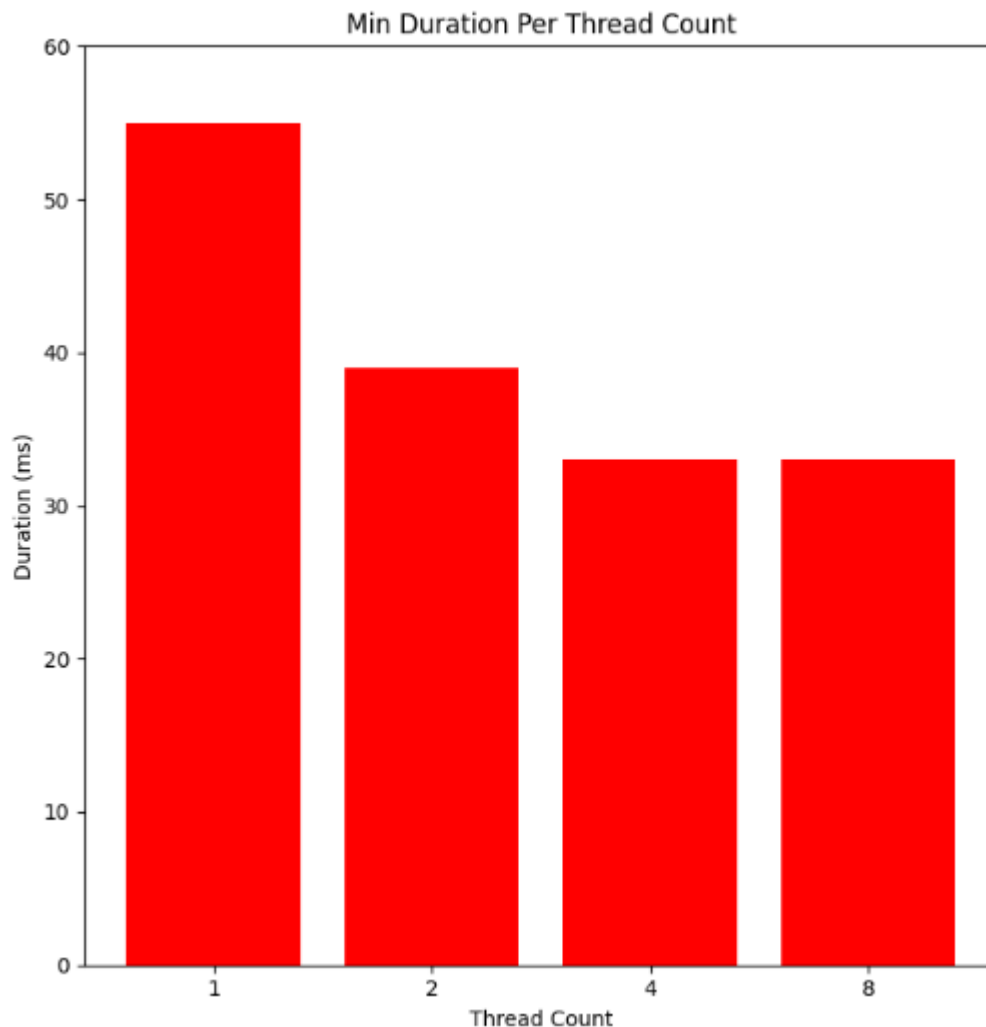
From this table you can see there is a clear increase in performance from one thread to two threads, the same for 2 threads to 4 threads and the same for 4 threads to 8 threads. There is a significant jump from 1 thread to 8 threads where the average duration for 1 thread is 68.56ms and the average duration for 8 threads is 35.66ms which shows a huge performance increase.

Plot Min durations for each thread count:

A bar plot using Matplotlib to display the performance metrics of a parallelized implementation of the "Wa-Tor" simulation with different thread counts. It showcases the minimum duration achieved for different thread counts in executing the simulation.

The purpose of the graph is to visually represent the minimum duration (in milliseconds) taken by the simulation to complete for different thread counts. Each bar represents the minimum duration obtained for a specific thread count. The x-axis denotes the number of threads used, while the y-axis represents the duration in milliseconds.

The graph provides for a comparison of the simulation's performance with different thread counts. Lower durations imply better performance or faster execution times. Thus, this graph helps in assessing the impact of parallelization (using different thread counts) on the simulation's runtime, aiming to determine the most efficient configuration for running the "Wa-Tor" simulation.

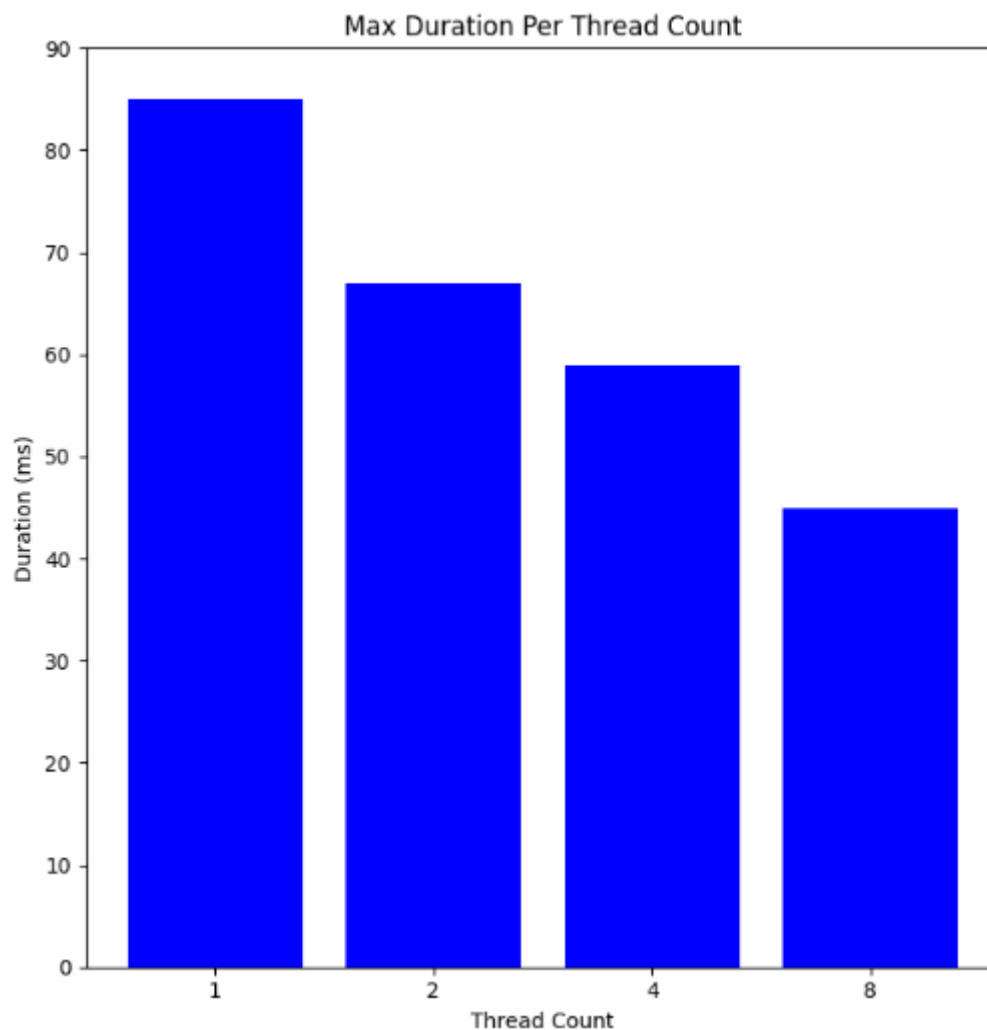


Plot Max durations for each thread count:

This bar plot using Matplotlib visualizes the maximum durations achieved during the execution of the "Wa-Tor" simulation with varying thread counts.

Like the previous plot, here each bar represents the maximum duration (in milliseconds) obtained for a specific thread count configuration. The x-axis displays the number of threads used, while the y-axis represents the duration in milliseconds. The purpose of this graph is different from the plot displaying minimum durations. It aims to illustrate the maximum time taken for the simulation to complete with different thread counts. Higher durations imply that the simulation took longer to finish its execution.

By visualizing the maximum durations across different thread counts, this graph helps identify scenarios where the simulation might experience spikes or higher execution times. These peaks can be crucial for assessing the simulation's performance under varying computational loads and optimizing the configuration for more consistent or efficient execution.

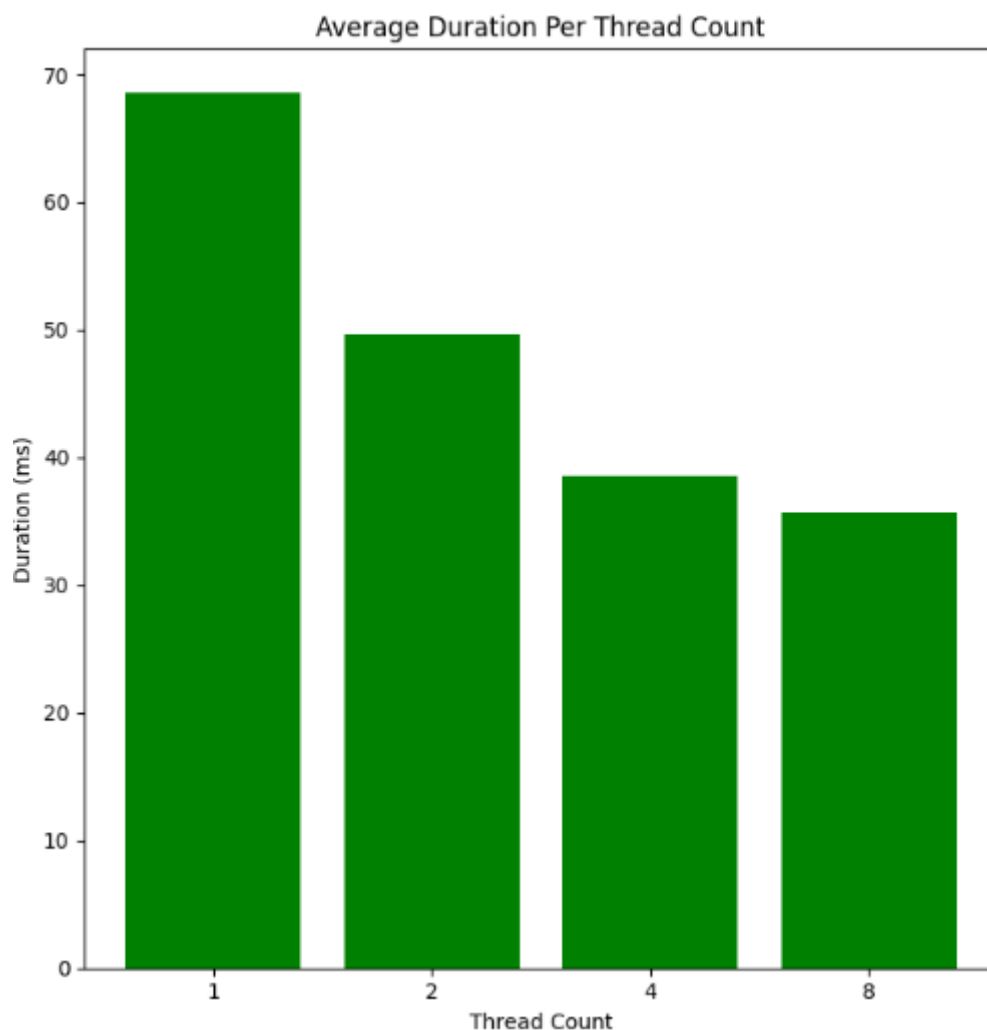


Plot Average durations for each thread count:

This bar plot using Matplotlib, specifically showcases the average durations achieved when running the "Wa-Tor" simulation with different thread counts.

Like the previous plots, this graph visualizes the average duration (in milliseconds) for each thread count configuration. Each bar represents the average time taken by the simulation to complete its execution for a specific thread count. The x-axis indicates the number of threads utilized, while the y-axis displays the duration in milliseconds.

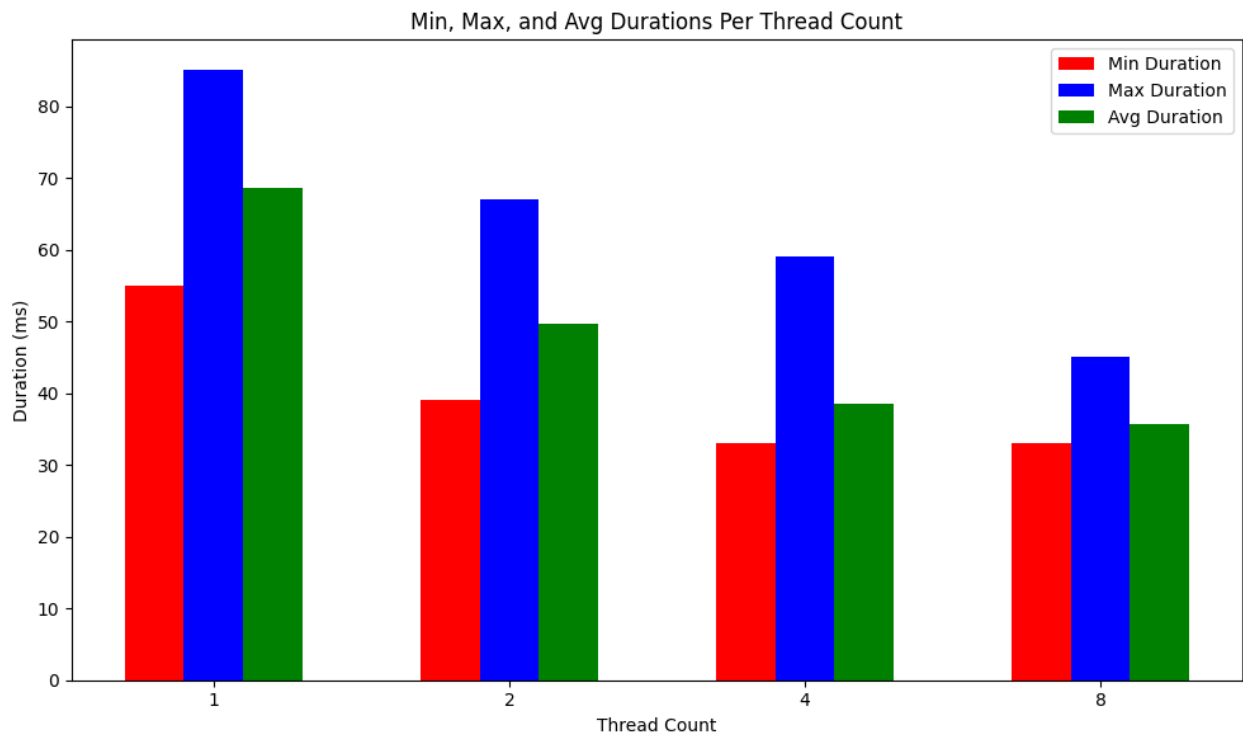
The primary purpose of this graph is to offer insight into the average performance of the simulation under different threading configurations. Unlike the minimum and maximum duration plots, which respectively represent the best- and worst-case scenarios in terms of execution time, the average duration plot provides a more balanced view of the simulation's runtime performance across various thread counts.



A subplot for Min, Max, and Average durations in one graph:

A bar plot using Matplotlib to showcase the minimum, maximum, and average durations achieved during the execution of the "Wa-Tor" simulation with varying thread counts.

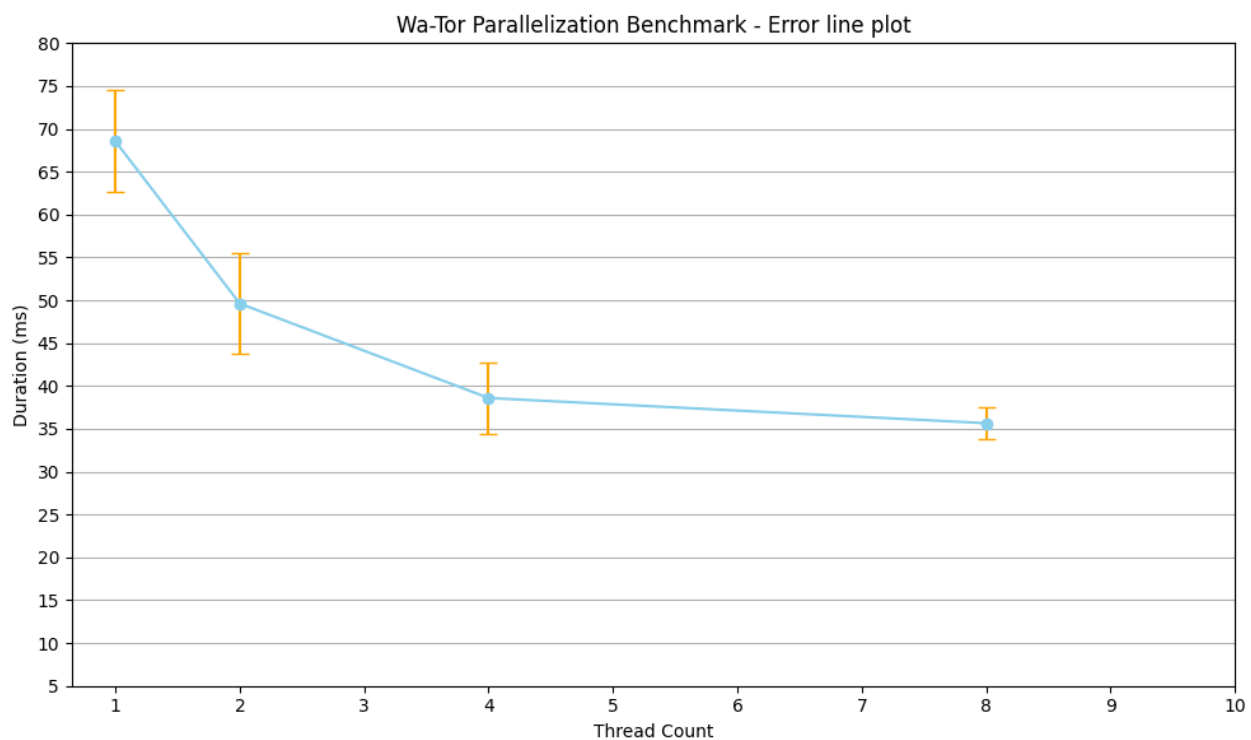
The purpose of this graph is to offer a comprehensive comparison of the minimum, maximum, and average durations for different thread counts in a single visualization.



Mean durations and standard deviations for each thread count:

I utilized Matplotlib to generate an error line plot, showcasing the mean durations and the variability (standard deviations) in durations observed for the "Wa-Tor" simulation across different thread counts.

The purpose of this error line plot is to visually represent both the average (mean) durations and the variability (standard deviations) associated with these durations across different thread counts. It allows for an understanding of the spread or dispersion of duration values around the mean for each thread count configuration.



Conclusion

The implementation successfully simulated the "Wa-Tor" ecological model using OpenMP for parallelization. The obtained duration results showcased variations in performance across different thread counts, providing insights into the simulation's scalability and efficiency. The utilization of multiple threads significantly influenced the overall execution time, demonstrating the impact of parallelization in computational simulations. The simulation exhibited distinct performance enhancements as the thread count increased. Moving from 1 thread to 2 threads demonstrated a notable reduction in execution time, showcasing a considerable speedup. Subsequently, scaling from 2 to 4 threads continued to reduce the duration of simulation steps, though with diminishing returns. Finally, transitioning to 8 threads resulted in further improvements, although the gains were less significant compared to previous increments.