Instituto Superior Técnico, Universidade de Lisboa
Master of Science in Computer Science and Engineering
Parallel and Distributed Computing

Parallelizing Game of Life: OpenMP Optimization and Performance Analysis

André Jesus (110860) André Páscoa (110817) Nyckollas Brandão (110893)

March, 2024

Abstract

This report presents the parallelization of the Game of Life using OpenMP. It discusses the approach, decomposition, synchronization concerns, load balancing, and performance results.

1 Introduction

Conway's Game of Life [1] is a classic cellular automaton that evolves based on simple rules governing the state of cells on a grid. The main motivation behind parallelization is to expedite the simulation process for larger grids and higher generations. In this report, we present our efforts to parallelize the Game of Life using OpenMP [3, 2], focusing on optimizing performance while maintaining correctness.

In Section 2, we provide an overview of the program, including its data structures and phases. In Section 3, we delve into our parallelization approach, discussing optimization techniques and decomposition strategies. Section 4 presents the performance results of our parallelized implementation compared to the serial version. Finally, in Section 5, we conclude our findings and discuss potential avenues for future enhancements.

2 Program Overview

2.1 Structures

The program employs key structures:

- **Grid**: A 3D array representing the cellular grid for the Game of Life.
- **Max Counts**: Array storing max counts of each species observed.
- **Max Generations**: Array storing corresponding generations of max counts.

2.2 Phases

The simulation progresses through phases:

1. **Initialization**: Allocates memory and initializes grid, species counts, and parameters.
2. **Cell Evolution**: Updates cell states based on Game of Life rules.
3. **Max Count Update**: Updates max counts and their generations based on grid's current state.
4. **Simulation Iteration**: Repeats cell evolution for specified generations.

3 Parallelization Approach using OpenMP

3.0.1 Modifications since Serial Version Delivery

Before beginning to explain the parallelization approach, it's important to note that in both the serial and OpenMP versions of our implementation, we made a simple modification to optimize the computation of neighbor indexes. Originally, we computed all three indexes (for x, y, and z) within the innermost loop, which resulted in inefficiency since the first two indexes could be calculated outside of the innermost loop. We modified the code to compute the indexes separately for each dimension, resulting in improved performance. Moreover, we replaced the modulo operator with conditional statements, which proved to be a faster alternative, contributing to the optimization of the index calculation process.

3.1 Approach Used

The parallelization approach primarily involves leveraging OpenMP directives to distribute the workload across multiple threads. OpenMP directives such as `parallel`, `for` were employed to parallelize the computation-intensive sections of the code.

To count the number of elements in each species in each generation, we used a `reduction` clause. Initially, we used just an `atomic` inside the loop, but we found that the reduction achieved better performance because it allowed for more efficient aggregation of results across threads without the locking necessary by the `atomic` directive.

The loop that iterates over the generations cannot be parallelized since each generation depends on the previous one. However, the inner three loops that iterate over each cell in the grid were optimized using a parallel `for` and a `reduction` clause to compute the number of elements for each species efficiently.

At the end of each generation, we have a loop with 10 iterations that updates the max counts for each species. Parallelizing this loop was not deemed necessary since it consisted of only 10 iterations, which did not justify the overhead of parallelization.

3.2 Decomposition Strategy

We adopted a straightforward decomposition strategy by parallelizing the main loops responsible for iterating over the grid cells. The grid is partitioned among

threads, with each thread handling a subset of cells.

At first, we implemented the partitioning with the `collapse(3)` directive, but this had worse performance due to cache locality. This decomposition strategy ensures that the workload is evenly distributed across threads.

3.3 Synchronization Concerns

Synchronization is crucial to ensure the correctness of the parallelized code. In our implementation, we encountered synchronization concerns primarily related to updating shared data structures such as species counts. We addressed these concerns by using appropriate synchronization mechanisms provided by OpenMP, such as `reduction` clauses; as previously mentioned, `atomic` was a working alternative that presented worse performance, yet also guaranteed the same behavior. These mechanisms ensure that concurrent accesses to shared data are properly coordinated, preventing race conditions and ensuring the consistency of the results.

4 Performance Results

The performance of the parallelized OpenMP version was evaluated against the serial implementation. The tests were performed in the machine 1 of lab 2, running GNU/Linux with 16GB of RAM and Intel(R) Core(TM) i5-11500 @ 2.70GHz Six Core (2 threads per core) with cache L1d (288 KiB), L1i(192 KiB), L2 (3MiB) and L3 (12 MiB). We ran the tests with 1, 2, and 4 threads since we didn't have access to a machine with more than 6 physical cores. We used the configuration examples that were made public in the project statement.

Table 1 presents the execution times for various input sizes and numbers of threads. The evolution of execution times in function of the number of threads, as depicted in Figure 1, correlates with the speedup trends showcased in Figure 2.

Configuration	**Threads**	**Serial (s)**	**OpenMP (s)**	**Speedup**
`life3d 1000 64 0.4 0`	1	34.6	36.2	0.96
	2	-	18.1	1.91
	4	-	9.2	3.76
`life3d 200 128 0.5 1000`	1	59.7	62.0	0.96
	2	-	31.1	1.92
	4	-	15.6	3.83
`life3d 10 512 0.4 0`	1	247.9	250.4	0.99
	2	-	125.2	1.98
	4	-	62.7	3.95
`life3d 3 1024 0.4 100`	1	628.0	635.5	0.99
	2	-	317.8	1.98
	4	-	159.0	3.95

Table 1: Execution times and speedup comparison for different numbers of threads/tasks and run configurations of the Game of Life.

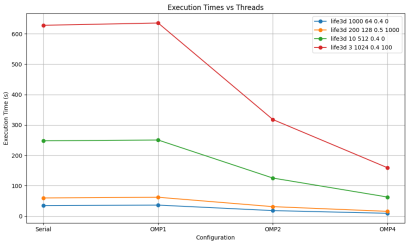

Figure 1: Evolution of execution times in function of no. threads.

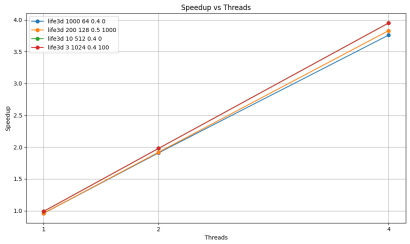

Figure 2: Evolution of speedups in function of no. threads.

Overall, the results indicate noticeable speedup as the number of threads increases, highlighting the effectiveness of parallelization. For instance, in the case of `life3d 1000 64 0.4 0`, the speedup from 1 to 4 threads is approximately 3.76, demonstrating substantial performance improvement.

The results demonstrate a trend towards increased speedup with the addition of threads, indicating effective parallelization. However, it's important to note that the relationship between the number of threads and speedup is not strictly linear. While doubling the number of threads generally leads to a noticeable reduction in execution time, the speedup does not scale perfectly proportionally. This deviation from linearity can be attributed to various factors, including overhead from thread creation and management, contention for shared resources, and limitations in hardware capabilities. Despite this deviation, the results consistently show improved performance with higher thread counts, underscoring the benefits of parallelization in optimizing computational tasks.

5 Conclusion

In conclusion, parallelizing the Game of Life using OpenMP has proved to be effective in accelerating the simulation process. By carefully designing the parallelization approach, addressing synchronization concerns, and optimizing load balancing, we achieved notable performance improvements compared to the serial implementation. Further optimizations and explorations could involve hybrid approaches combining MPI and OpenMP for distributed memory systems, as well as exploring more advanced parallelization techniques to achieve better scalability and performance.

References

[1] Conway's Game of Life - Wikipedia — en.wikipedia.org. `https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`. [Accessed 13-02-2024].

[2] Home - OpenMP — openmp.org. `https://www.openmp.org/`. [Accessed 26-02-2024].

[3] R. Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.