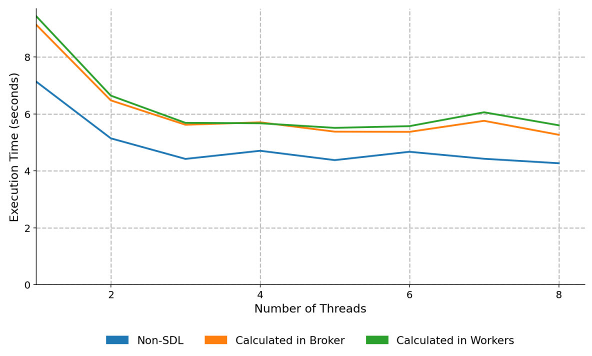

Figure 15: SDL Visualisation Implementations Comparison

As expected, both implementations of SDL are slower than the non-SDL implementation, due to greater communication overhead in the broker from additional RPC calls needed since the SDL requires knowledge of which cells are flipped.

5.5 Fault Tolerance

When running workers, they may fail due to unresponsive RPC calls, network issues or taking too long to respond. If a single worker fails, the broker, and thus the local controller will hang, as it waits for a response it will never receive. To handle these, the broker sends a request for each worker to compute its section inside a goroutine, and receives the response or an error on a buffered channel. If the worker sends an error or the timeout expires before a worker replies, we calculate the worker's section within the broker. To achieve this, we implemented a calculate next state function in the broker, which is called whenever an issue is detected with a particular worker, so the broker can calculate the next state of the worker's slice locally.

6 Distributed Optimisation

6.1 Parallel Workers

To optimise our distributed system, we parallelised the process of computing the next section for each worker by dividing the section assigned to each worker into smaller subsections and computing their next state on separate goroutines. We see from figure 16 that it scales well and is consistent with Amdahl's law, with both increasing threads and workers.

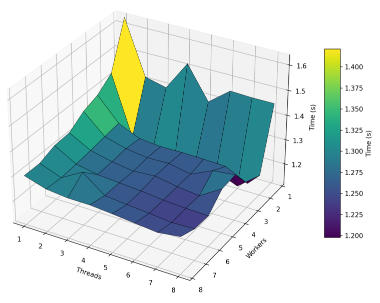

Figure 16: Threads vs Workers vs Time

6.2 Halo Exchange

To reduce communication overhead from RPC calls between the broker and workers we implemented halo exchange for our parallel worker implementation. Instead of sending each worker the entire world, we send only the cells in the worker's assigned section. When setting up connections to workers in the broker, we also signal each worker to dial its neighbouring workers, so it can make RPC calls to its neighbours its top and bottom rows.

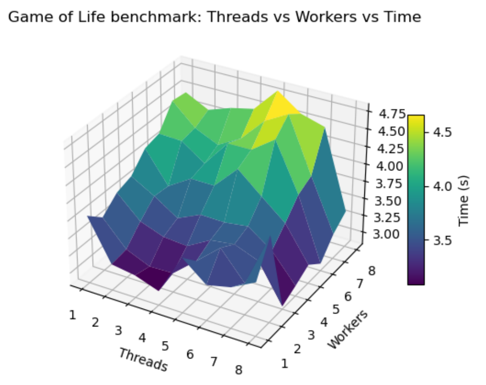

Figure 17: Threads vs Workers vs Time (Halo Exchange)

7 Conclusion and Potential Improvements

7.1 Conclusion

From our benchmarking and analysis, we learned that a significant portion of our scalability was limited due to communication overhead. In our concurrent version, we found that sending fewer, larger messages through channels was faster than sending many, smaller ones. Based on this finding, we reconstructed our implementation to build a version that uses full memory sharing. In our distributed version, scalibility was limited by high communication overhead. The cost of making RPC calls and merging results quickly outweighed the benefits of distribution. With the risk of one component failing, causing the entire network to fail, we successfully implemented fault tolerance to mitigate this issue, when a worker fails.

7.2 Potential Improvements

Some improvements we thought to implement for the distributed version next include redistributing sections of the world between remaining workers if a worker fails. For the concurrent version, We considered implementing the BPBC (Bitwise Parallel Bulk Computation) algorithm, for our concurrent version, which uses bitwise logic operations to calculate the next state of multiple cells in parallel.