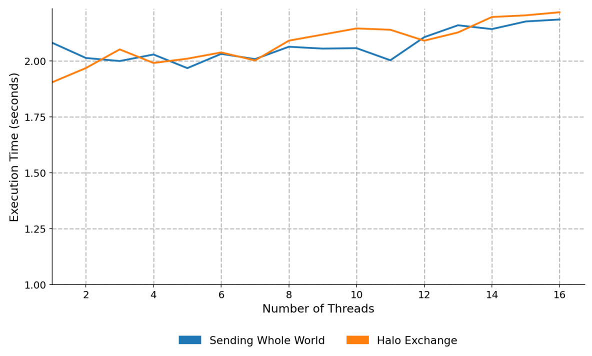

Figure 9: Sending Entire World vs Halo Exchange

As shown in Figure 9, the results were underwhelming, and the halo exchange implementation was marginally slower for a high worker count. However, this is not surprising - despite saving time by sending less data to workers, the two extra channel sends introduce additional communication overhead. Although these are parallelised, we can see that additional channel sends are far more costly than sending larger amounts of data in a single message. This result is similar to what was concluded in [4.2], and led us to construct an implementation that eliminates the communication overhead of sending back data through channels entirely by allocating two world slices in the main goroutine and passing a pointer to these slices to each worker. This incurred no race conditions as each worker read only from the first slice and wrote only to separate sections in the second slice. This implementation of using memory sharing instead of channels to communicate with workers, gave a moderate performance increase shown in Figure 10.

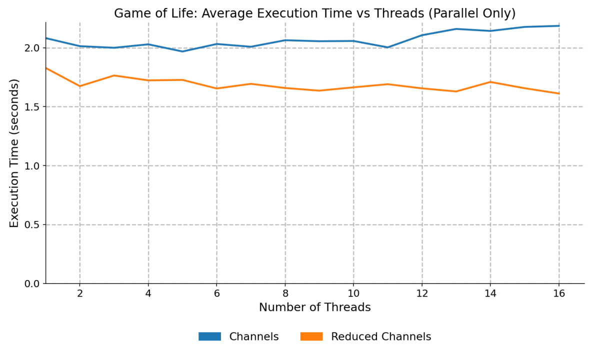

Figure 10: Workers Communication via Channels vs via Memory Sharing

4.4 Memory Sharing

We reconstructed our concurrent implementation to replace all channels with traditional synchronisation systems such as mutexes and condition variables. Our current optimised implementation already used memory sharing between workers and for storing the current world, so we expanded on this: we replaced each of the IO channels, the events and keypresses channel with condition variables with their own mutex locks used for accessing and setting each variable. This gave us a working version of the Game of Life using pure memory sharing. We also reconstructed our previous implementation to use full channels, without any memory sharing to compare performance. Figure 11 shows that full memory sharing performed faster whilst pure channels was slower compared to our previous mixed implementation.

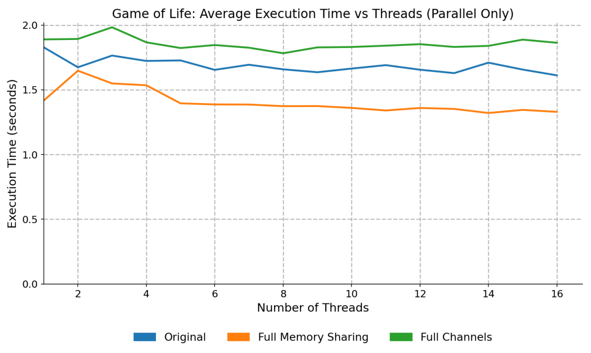

Figure 11: Pure Channels vs Original vs Pure Memory Sharing

4.5 Image Size Scalability

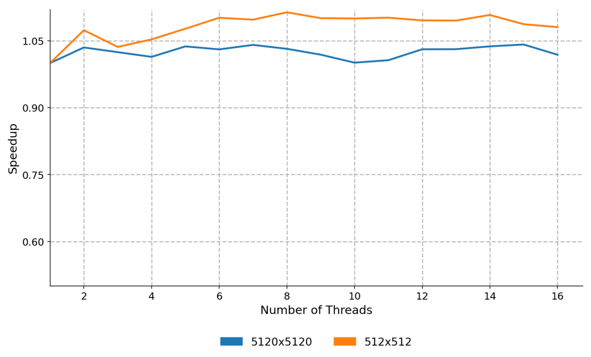

Figure 12: Scaling benchmark: 5120x5120 vs 512x512

Figure 12 illustrates how our implementation scales slightly better with a larger image size. This was to be expected, as with a larger image size, a higher percentage of total runtime comes from the workers, so a higher percentage of runtime is parallelised. Therefore, we speculate further improved scalability with larger image sizes.

5 Distributed Version

5.1 Functionality and Design

Our distributed implementation copied the function used by each worker to calculate the next world state from our concurrent implementation to a worker file. A broker is used to communicate with the client and workers using RPC calls. The broker and multiple workers were run on their own AWS EC2 instance. The local controller handles IO, keypress detection, getting alive cell count, and communicates with the broker. The local controller makes an RPC call to the broker to start the Game of Life. Each turn, the broker divides the world into sections based on the number of workers it is connected