

COMS20001- Game of Life Project

Functionality and Design

The aim of this coursework was to implement a working model of *The Game of Life*. We started by dividing the computation of different tasks and the grid (based on the number of workers), by implementing a “message passing model”, through exchanging messages in different channels, to prevent easily parallelizable tasks. For every send operation, there must be a matching receive. We used a Hard Channel (communication between different cores) to communicate between our distributor (in tile-0) and our worker threads in (tile-1) and a Soft Channel (communication using the same core) to exchange information between our LEDs, Buttons, Orientation, I/O streams to the distributor. This is to ensure that the worker threads work as efficiently as possible.

Synchronous Channels are used, as every input process continues when the corresponding output process on the same channel is ready (vice versa). In addition, it is exclusive to two threads (bidirectional). Using concurrent threads for our workers prevented the rules of the game to go through each cell sequentially. This also prevented the grid from returning back to the distributor sequentially by each worker, giving a significant increase in speed over sequentially running the program. Deadlocks were prevented by designing our program in such a way that we sequenced the tasks so that the processes don't wait too long for communication with each other and by following the Disjointness rules for channels and variables. Also, when a deadlock occurred, print statements were used to find which threads caused the blockage. In addition, Livelocks were prevented by ensuring that no threads were too busy responding to each other to resume work (Thread 1 does not act as a response to an action of thread 2, vice versa). Hence, this prevents the program from running into an infinite loop.

Parallel statements were used with the program body to (potentially) execute each block within the program body, on a free core concurrently. This is again, to increase efficiency over sequential execution. Finally, timer channels were implemented to display the processing time when the program is paused and to test efficiency, by varying the number of worker threads and grid size. The original timer had an issue, where it could not count beyond 42 seconds. This was resolved by storing the total time every 10 seconds. In the end, we have ensured that we have implemented explicit communication and parallelism between our tasks.

Tests and Experiments

16x16 image after two rounds for 2^n ($n \leq 3$) workers

Processing time for 2 rounds:

Trial	Time in ms	Average in ms
1	287	278.7
2	275	
3	274	



Processing time (in ms) for 100 iterations with 2 workers:

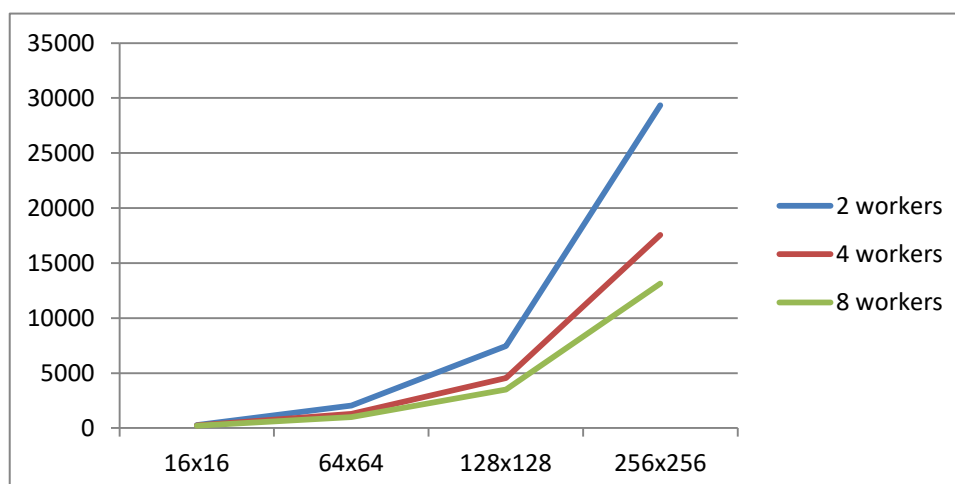
Trial	16x16	64x64	128x128	256x256
1	279	2044	7447	29437
2	280	2071	7480	29252
3	277	2056	7472	29354
Average	278.7	2057	7466.3	29347.7

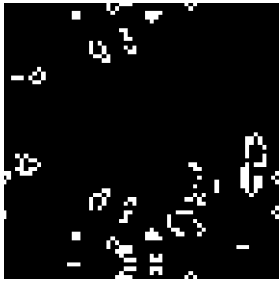
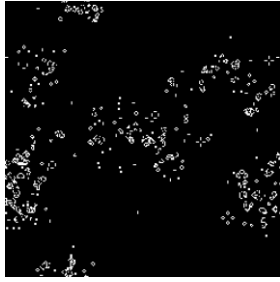
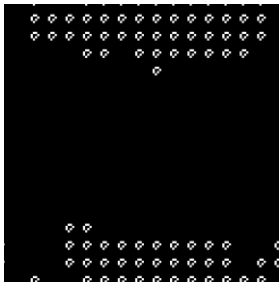
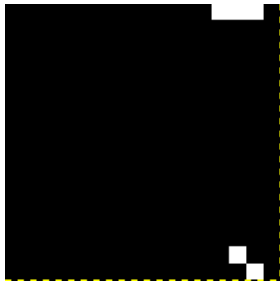
Processing time (in ms) for 100 iterations for 4 workers:

Trial	16x16	64x64	128x128	256x256
1	241	1299	4550	17562
2	238	1301	4553	17570
3	239	1302	4550	17572
Average	239.3	1300.7	4551	17568

Processing time (in ms) for 100 iterations for 8 workers:

Trial	16x16	64x64	128x128	256x256
1	229	1013	3505	13133
2	227	1014	3506	13136
3	228	1017	3503	13134
Average	228	1014.7	3504.7	13134.3



64x64		256x256	
128x128		16x16	

The tests were conducted without printf statements after every round, as this drastically increases processing time.

Variables used in our test:

1. Image size: Our results show that image size is proportional to the time to process it.
2. Number of Workers: The number of worker threads is inversely proportional to the processing time of the image.

Limitations:

1. Number of Cores : The number of logical cores, limited us to use a maximum of 8 worker threads
2. Storage space: The maximum memory on each tile is 256KB, an uncompressed image with resolution 512x512 or above requires memory greater than what is available on a single tile.
3. Number of workers must be a power of 2: Due to the way the grid is split between the workers, it is possible that it will be processed wrong if the number of workers is not a power of 2.

Critical Analysis

Our results show an almost exponential increase in processing time as the Image size increases. In addition, as worker count increases, processing speed increases, since most of the grid will be divided and processed faster with more workers. Therefore, for an image size $A \times B$, efficiency increases as the number of worker threads increases. However, the difference between the % of the decrease in processing time between images $2N \times 2N$ and $N \times N$ (where N is a power of 2) decreases. We could've improved this if we used more asynchronous channels. However, all our logical cores from both tiles were used up by the worker threads and synchronous channels. Furthermore, we could have shared cores to merge select statements of all combined functions per core into a single select statement. Also, we could have made the worker threads communicate with each other such that every even number of worker sends and every odd receives, ensuring no deadlock. Moreover, the maximum image we could process was 256×256 . This could've improved if we used bit packing. As it reduces storage by 8 times, thus memory efficiency increases. In conclusion, this project helped us understand the fundamentals of concurrent computing and improved our approach to performing quantitative analysis.

