XCore-200 Cellular Automaton Farm

Amar Khullar Zeping Chen
Computer Science
ak17050@my.bristol.ac.uk zc16388@my.bristol.ac.uk

Functionality and Design:

We have designed a program that correctly implements the game of life rules concurrently, across 4 worker threads. Our program can process image sizes up to and including 256x256. We are limited to this as we were not able to implement bit packing, so the memory required by the tile was too large for images of size 512x512 or above.

Our program is quite time efficient, we process 100 rounds on a 256x256 image in 49.58 seconds. We also conserved as much memory as possible, by only sending part of the image to each worker instead of the whole image. We send the rows required to operate on, and the one row above and below. Each worker then computes the rows allocated, and returns the output to the distributor where it is all put back together.

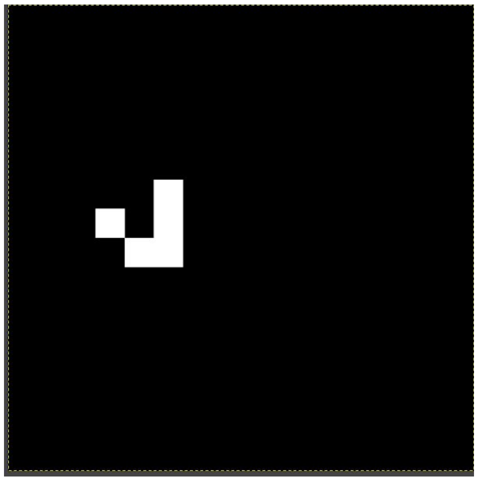
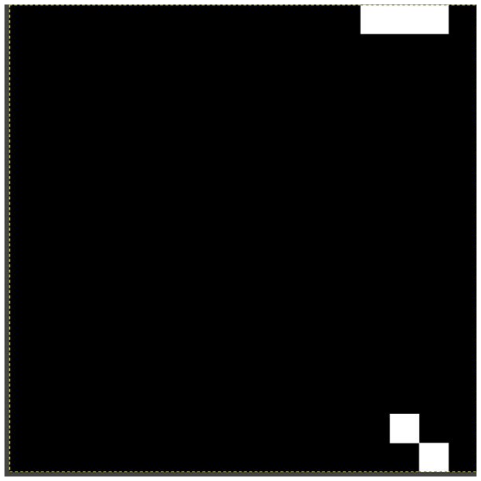
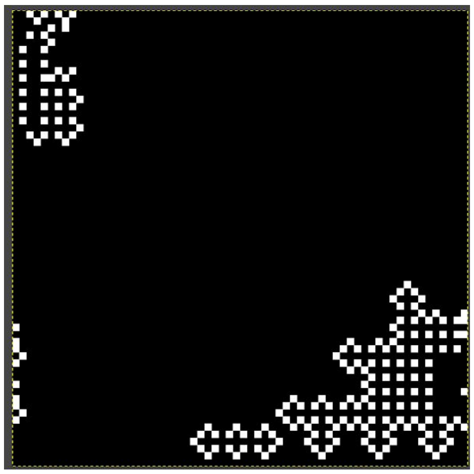
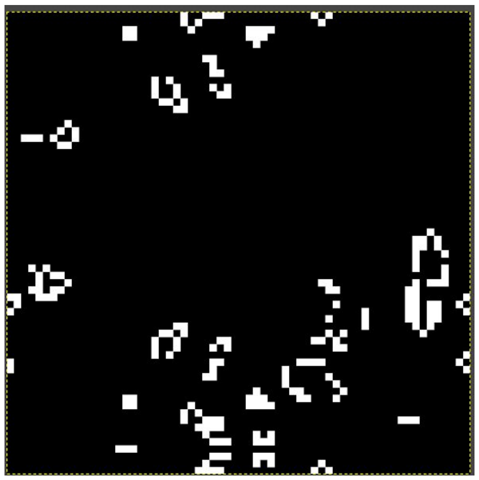
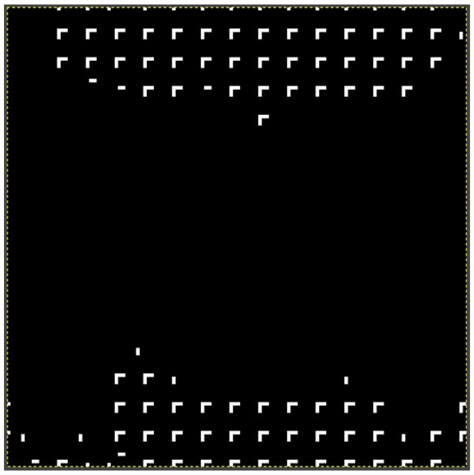
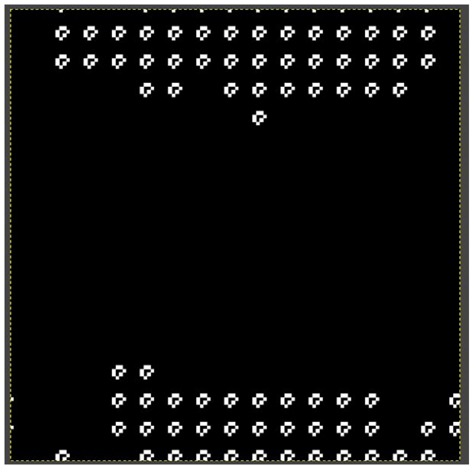
Our program solves the problem through a few different procedures. One of the main ones is FindNeighbours. FindNeighbours takes part of a grid, and a specific cell, and returns the number of alive neighbours using a for loop and comparisons. The adjust function is needed to implement the wrap around nature of the grid. Adjust takes an index and adjusts it to go to the other side of the grid when the index is at a boundary.

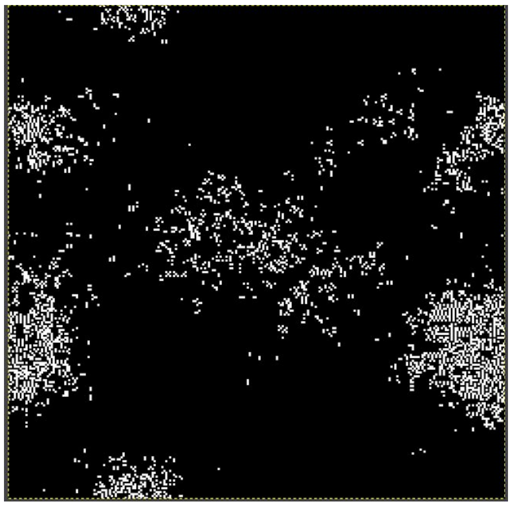
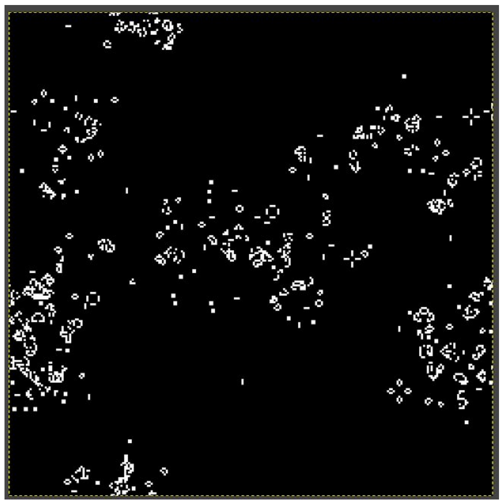
The worker function implements the main game of life functionality, along with distributor. It has a channel to distributor, which is how the grid cell values are passed to and from. The workers are given part of the grid each, then call the FindNeighbours function on each cell it's required to operate on, and calculates the new value for each cell. After all cells are calculated, it sends the relevant ones back to the distributor.

The distributor provides most of the functionality of the program. It communicates with the buttons, LEDS, input stream and the output stream. It also handles timing, pauses, and most importantly the distribution and collation of the cells to and from workers.

The buttonListener function handles button presses concurrently with the distributor.
The main function calls all of the required procedures and initialized the required channels.

Tests and Experiments:

Image	2 Rounds	100 Rounds
Test.pgm 14.23 s for 100 rounds		
64x64.pgm 15.85 s for 100 rounds		
128x128.pgm 23.70 s for 100 rounds		

256x256.pgm 49.58 s for 100 rounds		

When trying to process 512x512 images, we encountered a memory problem. The memory available to use on the tile the workers were on was 262144, the memory we used was 534320, which is roughly double. This is because our grid was not bit packed, bit packing would have saved a lot of memory, and would have allowed us to process much larger images.

We have also fully tested the buttons, LEDS and orientation sensor to see if they work as described in the assignment, which they do.

Virtues of our system:

- Efficient distribution and collation of cells to and from workers to allow for fast processing
- Fully functional buttons and LEDS
- Pauses when tilted and prints report giving grid data

Limitations:

- Cannot process images over 512x512 due to high memory usage as we didn't bit pack
- We only use 4 workers, implementing more workers may increase efficiency

Critical Analysis :

We are happy with our program, as it can correctly evolve an image according to the game of life rules for multiple rounds. We can process the 256x256 image in 49.58 seconds.

There are many things however that I would do different next time. We wasted a lot of time in the beginning of the project trying to implement it using interfaces, a way of programming that we were not very familiar with. We ended up scrapping our work and starting over, only using channels.

We also should have implemented bit packing and considered it earlier, we did not understand the importance of it. Also, it is harder to implement at the end, when all of your code is already written for unpacked arrays, than at the beginning. If we had more time we could have implemented bit packing and would be able to process images larger than 512x512.

Next time, we would spend more time planning our approach, and do more pair programming. Often one of us would be debugging for hours, and this could be prevented by working together better. There was a slight lack of communication, perhaps due to the language barrier between us. I believe we both spent well over 25 hours on this project, perhaps due to these circumstances.

In hindsight, I think we both underestimated the difficulty of this programming assignment. More theoretical knowledge prior to it, and better planning would have allowed us to make a better program.