COMS20001: Concurrent Computing Zack Dove zd17646@bristol.ac.uk

xCore-200 Cellular Automaton Farm

Andrei Bogdan ab17696@bristol.ac.uk

Functionality & Design

We have developed a concurrent implementation of Conway's Game of Life that uses multiple worker threads to process the data each round. The implementation applies the farmer worker pattern with our distributor function controlling the worker threads. The distributor receives the whole unprocessed image from the input file and splits it up into horizontal segments depending on the number of workers. The number of workers is dynamically allocated at run-time depending on the size of the image, this is discussed in further detail below. The worker receives the corresponding segment of the 2 dimensional world array and applies the game of life rules to compute the next game state. In our logic we had to pay close attention to the edge cells of the image; we had to make sure that the bottom of the image wraps to the top, left wraps to right and vice versa. The segment is then sent back to the distributor function which combines the segments from all the workers to produce the whole world.

To be able to process larger images (512x512 and above) we used bitpacking to reduce the size of our world array eightfold; this is done when the file is read in. Instead of having each a cell stored as an 8-bit unsigned char we packed 8 cells into one unsigned char, where each bit represents a cell. We had to alter our initial worker function so that the game of life logic is applied on a packed world array, using bitwise operations and manipulation. The world array is unpacked, printed and written to a file at the end of the iterations and when the SW2 button press signal is received.

To truly assess the performance of our implementation and understand whether adding certain features improves our processing speed we used a timer in our distributor function that provides accurate run-time values .XC's timer resets after roughly 42 seconds, therefore to be able to process larger images we had to implement our own timer logic that deals with this overflow. A feature that we added and assessed using the timer was the possibility to change between synchronous and asynchronous communication to the workers.

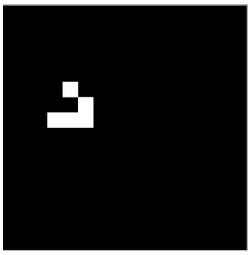
Our implementation makes use of the xCORE-200 eXplorer hardware features, requiring button input SW1 to start the processing, SW2 for exporting the current game state to an output file, alongside the the use of the orientation sensor to pause the game and print a report about the current game state. The report provides information on the number of alive cells, the iteration number and time elapsed after the image was read in.

We used the board's LED's to indicate the different stages of the simulation. Reading in is indicated by lighting the green LED, red is used to represent pausing the game and blue to indicate exporting to a file. Another separate green LED is used to show the ongoing process of the workers by flashing an alternating pattern. This is done by exchanging different values across a channel between the distributor and our LED manager function.

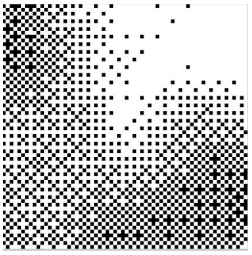
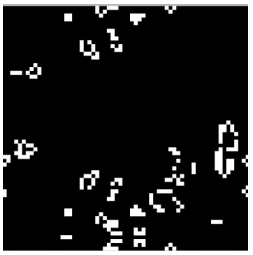
Tests & Experiments

The pairs of images below show the initial game state on the left and the result after 100 iterations of the simulation. The tests we're done on the provided test files and also on a 1024x1024 image which we created ourselves.

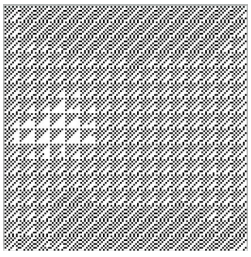
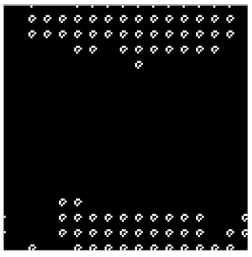
16x16:



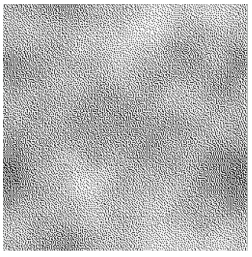
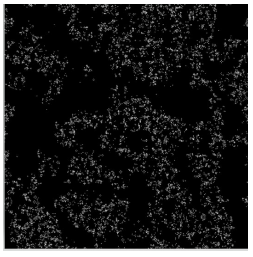

64x64

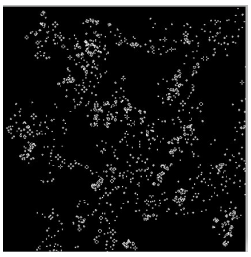
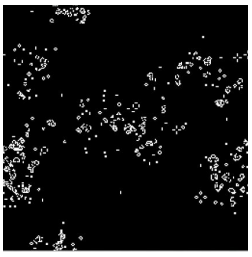
128x128:

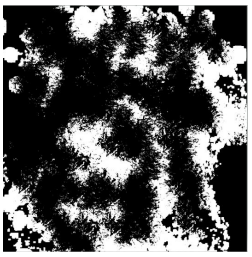
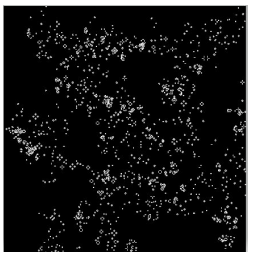
256x256:

512x512:

1024x1024:

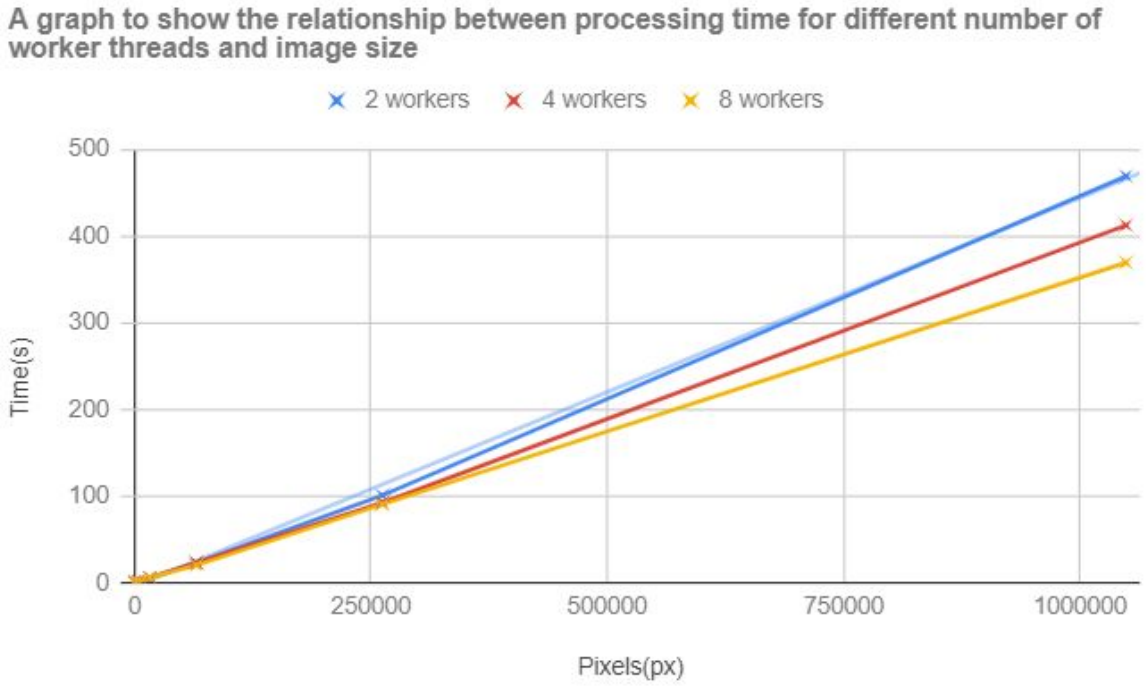



We carried a range of tests on our implementation to understand the factors for changes in performance. One of the first test was changing from synchronous to asynchronous channel communication between our distributor function and the workers. The table to the right shows the time taken to process 100 iterations for different image sizes for the different channel types with 4 workers. One can see that the time difference between the 2 channels is very small. In fact the asynchronous

IMAGE SIZE	SYNCHRONOUS	ASYNCHRONOUS
16x16	1.152546	1.151945
64x64	1.478043	1.163149
128x128	5.741722	5.067923
256x256	22.952284	22.095627
512x512	92.503213	89.456229
1024x1024	414.253793	358.891635

channel is only ever at most 1.3% faster than the normal channel. This slight improvement in performance might be explained by the possibility that the distributor will have to wait less time per iteration when receiving the data from the workers before sending them the next world state.

The second test that we did was changing the number of worker threads. The graph below shows the relationship between the number of pixels in the input file and the time taken to process 100 iterations for images up to 1024x1024.



It is clear from the different gradients that the more concurrent the system becomes the more pixels processed per second. For smaller images, one can see that the time difference between the 3 systems is rather small since the lines get closer together as the number of pixels decreases. In fact our results showed that for images of size 128x128 and below, the 4 worker implementation can be faster than the 8 workers. For example for images of size 128x128 the average time taken to complete 100 iterations in a system with 4 workers is 5.732788s compared to 6.067641s for a system with 8. We think that this was because for smaller images our distributor function might longer to piece back the 'world' matrix since there are more segments collected from the 8 workers compared to the 4 and since this is done every iteration, the small time difference accumulates. This small difference becomes irrelevant for bigger images since the 8 worker implementation is, on average, 43.144364 seconds faster than the 4 worker implementation. Since discovering these results we altered our implementation so that the number of workers is decided at run-time; 4 workers for images of size 128x128 and below and 8 workers otherwise. This test was repeated 3 times for each image size to make sure our results are accurate and reliable. All the data used can be found in our 'timings' file attached to this project.

To ensure our experiment was fair we assessed the performance of our implementation based on different types of images. We wanted to see if the initial game state affects the performance in any way. The table to the right below show the time take to process different images of size 128x128 with 8 workers.

The different images are as follows: A- white , B-black, C-primarily white with some black, D-primarily black with some white and the original test file. The biggest time difference is roughly 30ms between any image. This shows that the initial game state is less significant than any other factors that we tested. The purpose of this test was to provide inclusive results by testing different types of images but we came to the conclusion that the impact it has on the performance is too small to be regarded.

DIFFERENT 128x128 IMAGES	TIME(s)
IMAGE A	6.046013
IMAGE B	6.053616
IMAGE C	6.069821
IMAGE D	6.081024
ORIGINAL	6.069021

Critical Analysis

The maximal size image our system can process is 1328x1328 with 8 workers. With regards to speed our system can process an image of size 512x512 in 90.38s which on average means that the time to process one pixel per iteration is 0.0096ms. We realised that this is not particularly fast when comparing results with other teams, however when looking at maximal input size our implementation often outperformed others.

We concluded that our system can be optimised further, but efficiently stores the state of the game. One logic optimisation would be for the distributor to piece the world back together at the end of the iterations or when the ‘SW2’ button is pressed rather than doing it after every iteration, reducing the amount of data passed through the channels.

Furthermore we could have taken advantage of the time taken for an image to be read in by getting the workers to start work as soon as n rows are read in. For example once 25 rows are read in get the 8 worker to start applying the Game of Life rules and repeat this until the whole image is processed.

To make our tests more complete we could have experimented with different ways of splitting up the image . Our approach was to split it horizontally relative to the height of the image and number of workers. Perhaps a vertical approach or splitting the image into quadrants may have produced faster results.

A functionality provided by XC that might have improved our communication between worker and distributor is the use of client-server interfaces, however we quickly decided against it as interfaces only allow communication in one direction, increasing complexity for worker distributor communication.