COMS20001 - Cellular Automation Farm

Varadhan Kalidasan (vk14831) and Jonathan Wingrove (jw14897)

Functionality and Design

We designed and implemented three main methods for the cellular automation farm. Each of which had its own strengths and weaknesses in term of speed and memory consumption. Each design implemented the division of work between worker threads in different ways.

1. Our first method was to divide the grid containing the image into 8 sections and pass each worker a section of the grid to iterate over before passing that section of image back to the distributor where the image would be reconstructed. This system worked well, and was very fast, as little communication is needed between the distributor and workers during each iteration. However, this system also took up a lot of memory, as each worker was being passed a large section of the image and so it was not possible to process large images on this implementation.

2. Our second method involved each worker processing a single line, then passing it to a harvester which would store all the processed lines. The distributor would then send each worker a new line to process, when the entire image has been processed the harvester then sends the fully processed image to the distributor. This system was very effective as it meant each worker was only being sent a small section of the image this in turn means that each worker takes up much less memory allowing us to process very large images. However, this system is slower than method 1 as each worker is only processing a single line at a time, so there is a lot of communication during each iteration between the distributor, harvester and workers.

3. Our third method was similar to method 2, however, it involved splitting a single line of the image into 8 sections and each worker processing a single section of the line. This meant that the workers took up even less memory, in theory allowing us to process even larger images. However, in practice this was not the case, this is because at this size the distributor which stores the image being processed became too large to fit on a single core, so although there was excess memory on the other core it couldn't be used. This method was also the slowest

by far as only very small parts of the image were being sent to each worker at a time, this meant lots of communication between the harvester, distributor and workers occurred every iteration.

In the end we decided method 2 was the correct trade off between speed and memory consumption, and was therefore the most efficient and effective way to solve the problem.

Other Design Choices

In order to process sections of the image on different workers we would pass each worker the section it is meant to process plus an extra row and column on all sides, this allowed the worker to process that section of the image, without the need for the whole image to be passed to the worker.

We decided to reconstruct our image on a separate harvester thread instead of on the distributor thread, this was to avoid storing two large images on the same thread. Allowing us to put each thread on a different core for a more memory efficient arrangement.

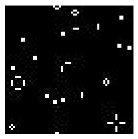
We compressed each byte of the image into a single bit, dramatically reducing the space taken up by the image. We also chose to store those bits in unsigned shorts, instead of unsigned characters, this was originally to reduce the number of communications needed between threads as you would be sending twice the data in one communication, however, it also reduced the amount of memory the stored image needed.

Tests and Experiments

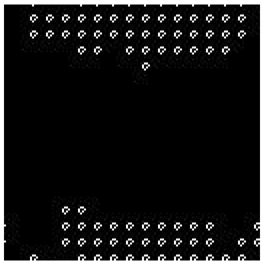
We have tested our design against multiple image sizes as we increased the maximum image size we were able to process. We mainly used the 128x128 example image to test for correctness of processing as this image always produces a stable output after the first few rounds. We also tested against single still life images such as the block, beehive and loaf.

16x16 After 250000 rounds.

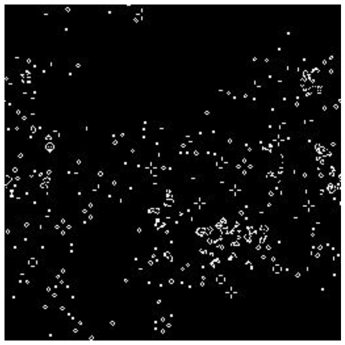
64x64 after 37629 rounds.



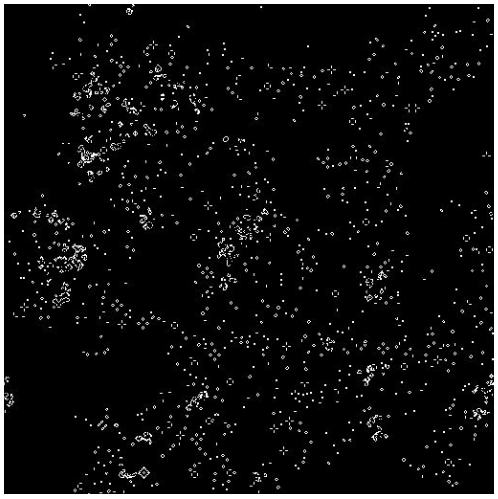
128x128 after 8300 rounds.



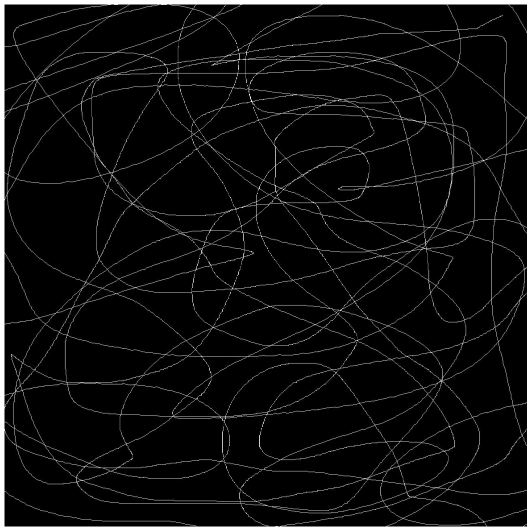
256x256 after 2884 rounds.



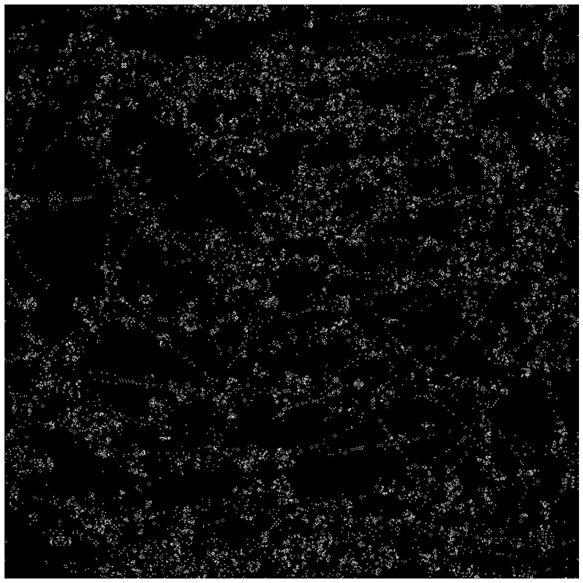
512x512 after 697 rounds.



1296x1296 image before processing.



1296x1296 after 177 rounds.



Critical Analysis

The maximum image size that we were able to process is 1296x1296, an image of this size uses all of the memory available on one core and most of the memory available on the other to store and process. We may have been able to improve upon this image size by splitting the storage of the full image between multiple cores, this would have allowed us to equalise the memory usage between the cores.

The processing speed is dependant on the size of the image, as shown in the table below.

Image Size	Rounds per second
16x16	9259
64x64	738
128x128	184
256x256	115
512x512	18
1296x1296	5

Increasing image sizes also affect the speed at which the image can be read in and written out with the 1296x1296 image taking around 23 minutes to read/write, we believe that this is mainly due to the raw size of the image rather than the compression we use to store it.

Our program also requires the image size to be manually entered when the input image is changed. As the image size is included in the file header we could have utilised this to improve our code.