COMS20001 - Concurrent Computing - Coursework 1

map ("Adam " ++) ["Beddoe", "Fox"] — ab16301/af16371

1 Functionality & Design

Our xCore-200 eXplorerKIT implementation of the John Horton Conway's Game-of-Life correctly implements the following functionality:

- Correctly evolving Game-of-Life, according to the given rules
- Multiple worker threads for evolution
- Button, Board orientation and LED behaviour
- Ability to process larger than 512x512 images
- Memory on both tiles
- Timers to measure processing speed
- Comprehensive unit testing
- Ability to generate random images up to 1344x1344 on the board

In order to maximise the speed and memory efficiency of our implementation we split the incoming image up across a number of threads, either 2,4 or 8. Each of these threads receives an equal column of the total image. e.g. with a 64x64 image and 2 threads, each thread receives at 32x64 portion of the image.

1.1 Worker Types

We then have a few differing worker implementations each with their own benefits and drawbacks:

1.1.1 Unpacked Worker

This was the first worker we implemented, and is therefore the simplest. It reads in the segment of the image it is passed, before passing the leftmost column to the thread on its left and vice versa for the right column, in order to correctly calculate the number of neighbours a cell has. Having been passed a channel to these two. It will then continue iterating and communicating with the adjacent threads until told to send it's current state back to the Distributor function in main.xc.

1.1.2 Packed Chunk Worker

As the name suggests, this worker differs in two main ways from the one above: we use bit packing to reduce the size, and we split the image up even further into chunks. We define a 'packedChunk' as:

```
struct packedChunk {
    uchar left, right, top, bottom, corners;
    uchar row[8];
};
```

Packing into an unsigned char (uchar) we can store 8 values in each, drastically reducing the memory required. The one-dimensional array *row* stores 8 rows of 8 values, thus storing 64 values per chunk. The *left, right, top, bottom* variables, hold the corresponding row of the neighbouring chunk, e.g. left stores the rightmost column of the chunk to the left. Additionally, we use the 4 least significant bits in *corners* to store the 4 adjacent corners.

1.1.3 Packed Column Worker

Our final worker is the amalgam of the previous two. Utilising the bit packing functions created for *Packed Chunk Worker* and the column passing paradigm of *Unpacked Worker*.

2 Tests & Experiments

2.1 2 Round Images

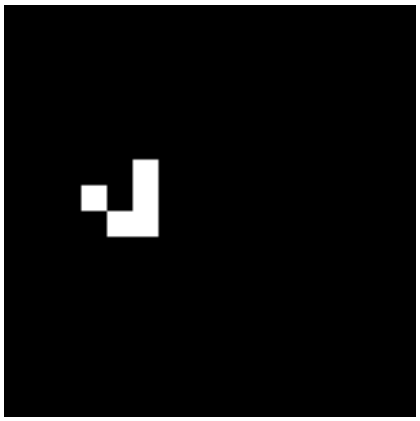

Figure 1: 16x16

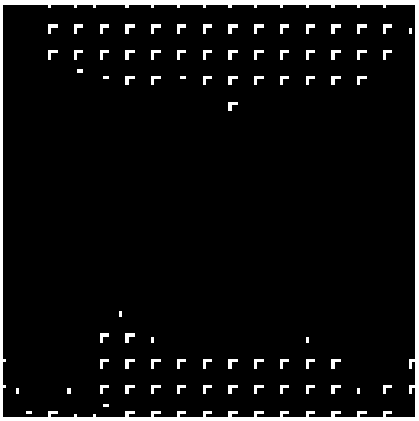

Figure 3: 128x128

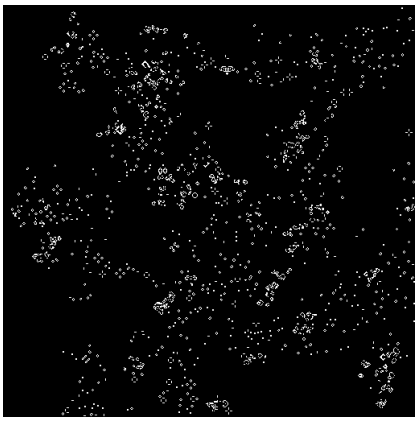

Figure 5: 512x512

Figure 2: 64x64

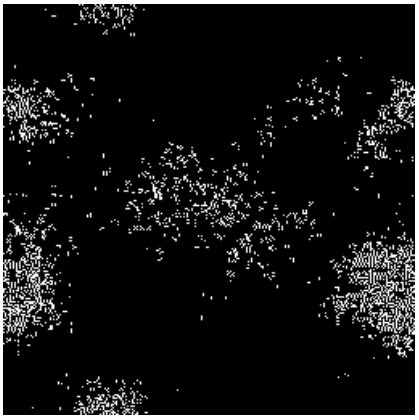

Figure 4: 256x256

3 Critical Analysis