**1. Functionality and Design (1 page max)**

For efficient processing of larger images, the input image needs to be split into sections and passed onto workers. We represented the whole board as a 3D array of the following shape, where extra two spaces are added for the ‘ghost rows’.

uchar board[NUM\_WORKERS][FARMHT + 2][IMWD]

Ghost rows are used for reference by workers but are not processed upon. At the time of writing we are looking at implementing a feature to allow workers to communicate their top and bottom rows to each other upon request.

To allow for that kind of flexibility we preferred interfaces as the primary data exchange strategy. The key feature is the extending API which enabled us to safely add on bitpacking and experiment with other features without touching the core implementation.

At a certain point the system ran out of memory even before reading in the whole image (see section 2). We overcame this obstacle via bitpacking:

uchar result = 0;

for (int bit = 0; bit < BYTE\_SIZE; bit++){

c\_in :> inputVal;

result |= (inputVal==255) <<(7 - bit);

}

board[workerId][y][x] = result;

This reduces the size of data passed around by the factor of 8, resulting in noticeable processing improvements.

While the system required constants, we tried to keep the core games’ (processing) logic agnostic to the rest of the implementation. This allowed us to experiment with the system more freely (see section 2). At the time of writing there is not a concrete unit test suite implementation, mainly due to the lack of research into testing techniques in C and xC.

*(TO BE CONTINUED)*

*For a mark up to 70 start to experiment with different system parameters (e.g. number of worker threads, data exchange strategy, synchronous vs. asynchronous channels etc) and draw conclusions about the performance of your system in your report. Analyse which system parts for processing/communication are limiting factors. For a 1st class mark you should present/submit a system, which can process large images fast; and you should show a very good understanding of the concurrency concepts relevant to your code.*

**I/O**

**2. Tests and Experiments (2 pages max)**

*Show the result of the given 16x16 image after 2 rounds. Describe briefly the other experiments you carried out, provide a selection of appropriate results and output images. This must be done for at least the example images provided and for at least one example image of your own choosing (showcasing the merit of your system). List the important factors responsible for virtues and limitations of your system.*

All processing times are for 100 rounds of image processing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Board Size | Processing Time (1 worker) | Processing Time (2 workers) | Processing Time (4 workers) | Processing Time (8 workers) |
| 16x16 |  |  |  |  |
| 64x64 |  |  |  |  |
| 128x128 |  |  |  |  |
| 512x512 |  |  |  |  |
| 1000x1000 |  |  |  |  |

**16x16 Image Testing (2 rounds)**

**64x64 Image Output (2 rounds)**

**128x128 Image Output (2 rounds)**

**512x512 Image Testing (2 rounds)**

**3. Critical Analysis (1 page max)**

Discuss the performance of your program with reference to the results obtained and indicate ways in which it might be improved. State clearly what maximal size of image your system can process and how fast your system can evolve the Game of Life. (Make sure your team’s names, course, and email addresses appears on page 1 of the report.)

Shortcomings:

1. Testing
2. Components not decoupled enough (often have to go change things all over the code)
3. Not utilising [[combinable]]

use to have in the distributor looping over the board but now we need a function call to somewhere else calculate the next value

We needed more foresight into variables and planning with variables because things stopped being squares when ghost rows were added

Declared all workers and changed the amount we used