Concurrent Report

Ainsley Rutterford
ar16478@my.bristol.ac.uk
Computer Science

Harry Waugh
hw16470@my.bristol.ac.uk
Computer Science

November 29, 2017

1 Functionality and Design

Our system currently uses up to 8 workers to evolve the Game-of-Life repeatedly. Our system is deadlock-free, implements the correct button, board orientation, and LED behaviour, and can process images up to 1264x1264 pixels using memory on both tiles.

The biggest problem we encountered was trying to run images bigger than 512x512. Originally we were reading the bits from the .pgm file into a 2 dimensional array of unsigned chars, before 'packing' this array into an array of unsigned chars that represented each pixel by a bit instead of a byte. This packing process allowed us to save space as we only had to store an array which was 1/8th the size of the original array. We proccessed the Game-of-Life on this array. Eventually we had to free even more space in order to run larger images. To do this, we changed the dataInStream function, so that it read the pixel values straight into the smaller array, representing each pixel as a single bit. At this point we could process images up to 688x688. We then then realised that in `dataInStream`, we didn't have to store an unsigned char array at all. We simply processed each byte and sent each byte to the distributer as we read from the file. This meant that instead of storing an entire array of unsigned chars of size [ImageHeight][ImageWidth / 8], we stored a single unsigned char.

We also encountered problems while implementing a timer. The first problem was that the timer seemed to overflow roughly every 42 seconds. We decided to create a function that would compare two times given, and would return whether or not the timer has overflowed. We would then call this function when the board was tilted, and if the function returned true, we would increment a counter which counted how many times the timer has overflowed. The time displayed would be the timers current value added to the overflow value multiplied by the counter value. Using this method, our clock no longer overflowed.

Early on in the development of our system, our workers functioned differently. Each worker would work on a single byte of data. We would send each worker a 3x3 array of unsigned chars, with the middle char being the char that the worker would work on. The worker would then send only the completed char back. We realised that this was very inefficient as each worker required 8 extra bytes to work on a single byte. Image processing was very slow using this method so we decided to update our workers to work on a whole strip of an image at once. This meant that for whole chunk of an image to be worked on, only an extra strip of pixels at the top and bottom of the strip had to be sent as the sides of the image wrap round to each other. Using this new worker strategy, our image processing was up to 11 times faster.

2 How to create a bulleted list

This is the Second Section. Let's also see if this text wraps. Will this be below or beside Introduction? Here is how to create a list:

- First bullet point
- This is the second bullet point.
- And so on...

Then we can carry on the section. But we must be careful because the beginning of this line will be indented as it is after a list.

3 New Section

This is a new Section. This one should wrap too. This is the third section.