Omran Alhaddad - oa17248
Yash Benchmark - ya17227

COMS20001 Concurrent Computing (2018-19)

Cellular Automaton Farm

Functionality and Design:

The cellular automaton was implemented on XMOS xCore-200 board using multi-thread workers. The board's buttons, orientation sensors, and LEDs were enabled to control and visualise the aspects of the game. The game was initialised from a PGM image “test.pgm” and exported as PGM image too “testout.pgm”.

The program starts reading and processing of an image by pressing button SW1 and storing the incoming pixel on multi-thread workers (where the number of workers is declare at the beginning of the code), indicate starting and reading by lighting the green LED, and indicate ongoing processing by flashing of the other separate green LED (on/off). The game triggers the export of the current game state to image file by clicking on the button SW2 and the blue LED explored on the xCore-200 board while exporting to PGM file. The program paused by tilting the board 30 or more degree, indicate by lighting the red LED, also on the console it prints out how much its tilted and the processing time elapsed in addition to the current number of rounds and live cells, the board continue processing once it back to its normal status again.

Figure 1 explains the relationship between the threads on the board. By pressing SW1, the ‘distributor’ thread receives a stream of pixel values from ‘DataInStream’. Then equally, it sends these pixels to a number of worker threads to operate on different parts of the image in parallel. Each worker sends/receives two lines to/from its neighbors via ‘distributor’ thread, for example ‘worker 0’ sends ‘line 0’ to ‘worker 7’ and receive of it ‘line 63’ meanwhile ‘worker 0’ sends ‘line 7’ to ‘worker 1’ and receive ‘line 8’ (the figure 2 clarify the communications).

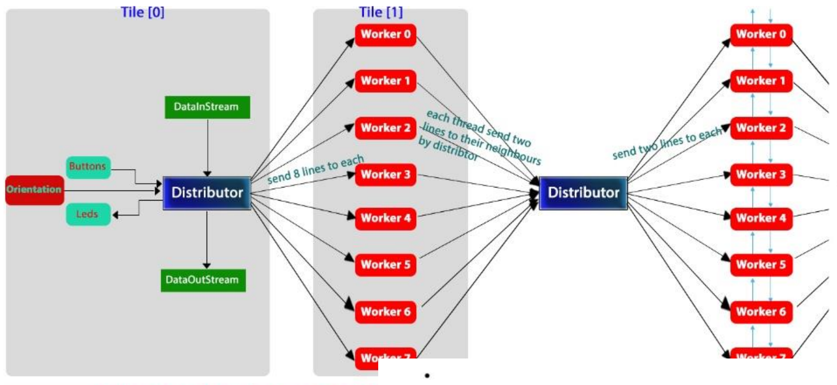


Figure 1: Workflow between thread

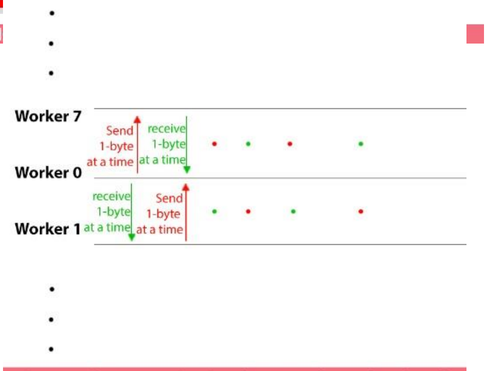


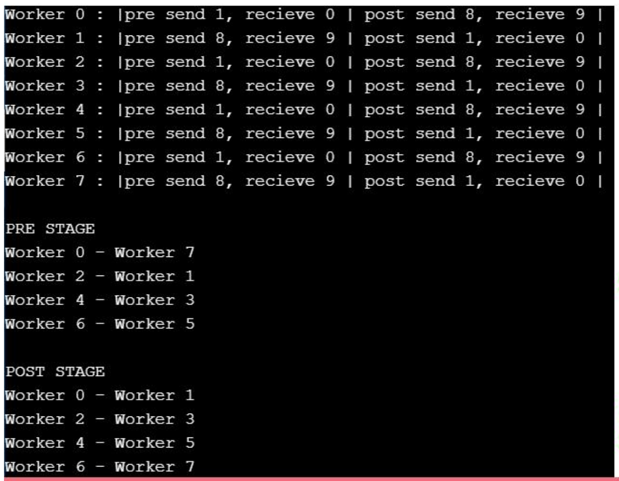
Figure 2: the communications between the worker threads

CODE OPTIMIZATION TECHNIQUES:

1- - Storing array only in the worker threads function (not in the ‘distributor’ function);
   - the two arrays are of size **[** (IMHT/W) + 2**]** **[** IMWD**]**.
2- - Works communicate synchronously via ‘distributor’;
   - they send and receive neighbor lines where each worker thread send and receive two lines.

```
 int flag = 0, round = 0, pre, post, coeff;                 /* into worker function */

pre = (Wnum % 2 == 0) ? 1 : (IMHT/W);
post = (Wnum % 2 == 0) ? (IMHT/W) : 1;
coeff = (Wnum % 2 == 0) ? 1 : -1 ;


fromDist :> flag;  //recieve a signal from Distributor to start exchanging neighbour pixels

//PRE STAGE
for(int i = 0; i < IMWD; i++)
{
  fromDist <: arr1[pre][i];//send
  fromDist :> arr1[pre-coeff][i];//recieve
}
//printf("\nWorker %d",Wnum);  //print worker number

//POST STAGE
for(int i = 0; i < IMWD; i++)
{
  fromDist <: arr1[post][i];
  fromDist :> arr1[post+coeff][i];
}
fromDist <: flag;  //send a signal to Distributor after the end of exchange
```

```
                  /* into distributor function */

//Workers communicating synchronously to recieve modified neighbours
  for(int i=0;i<W;i++)
  c_worker[i] <: button;
  //1.sending flag to workers to start communication
  //2.also sends the signal whether to export image or not
  for( int i = 0; i < W; i+=2)
  {
    //PRE STAGE
    for( int j = 0; j < IMWD; j++)
    {
      c_worker[MODWORK(i-1)] :> pix;
      c_worker[i] :> pix2;
      c_worker[MODWORK(i-1)] <: pix2;
      c_worker[i] <: pix;
    }
  }
  for( int i = 0; i < W; i+=2)
  {
    //POST STAGE
    for( int j = 0; j < IMWD; j++)
    {
      c_worker[MODWORK(i+1)] :> pix;
      c_worker[i] :> pix2;
      c_worker[MODWORK(i+1)] <: pix2;
      c_worker[i] <: pix;
    }
  }
  for(int i=0;i<W;i++)
  c_worker[i] :> liveCells; // recieving flag from worker
```



```
Worker 0 : |pre send 1, recieve 0 | post send 8, recieve 9 |
Worker 1 : |pre send 8, recieve 9 | post send 1, recieve 0 |
Worker 2 : |pre send 1, recieve 0 | post send 8, recieve 9 |
Worker 3 : |pre send 8, recieve 9 | post send 1, recieve 0 |
Worker 4 : |pre send 1, recieve 0 | post send 8, recieve 9 |
Worker 5 : |pre send 8, recieve 9 | post send 1, recieve 0 |
Worker 6 : |pre send 1, recieve 0 | post send 8, recieve 9 |
Worker 7 : |pre send 8, recieve 9 | post send 1, recieve 0 |

PRE STAGE
Worker 0 - Worker 7
Worker 2 - Worker 1
Worker 4 - Worker 3
Worker 6 - Worker 5

POST STAGE
Worker 0 - Worker 1
Worker 2 - Worker 3
Worker 4 - Worker 5
Worker 6 - Worker 7
```

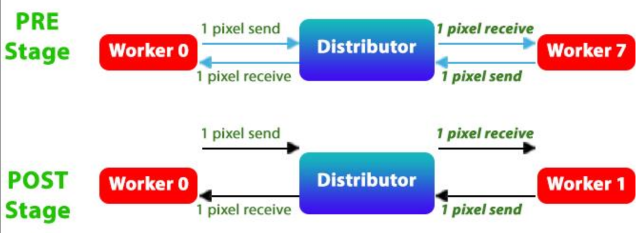


Figure 3: The synchronization between 'worker threads'

Tests and Experiments

The program was performed on different input image sizes over 2 and 100 rounds shown in Figures 2,3,4 and 5.

The tables below show the processing speed for multiple image sizes for eight and four workers:

The times measured in ms Using 8 Workers		
Image Size	**after round 2**	**after round 100**
16x16	447	28911
64x64	346	17112
128x128	364	17091
256x256	499	23629

The times measured in ms Using 4 Workers		
Image Size	**after round 2**	**after round 100**
16x16	341	17099
64x64	349	17051
128x128	400	17165
256x256	563	24159

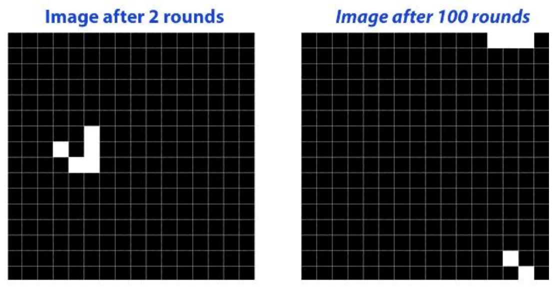


Figure 4: 16x16 image

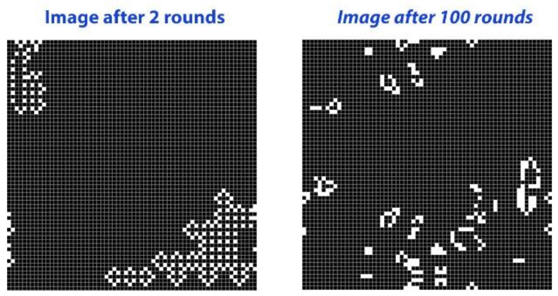


Figure 5: 64x64 image

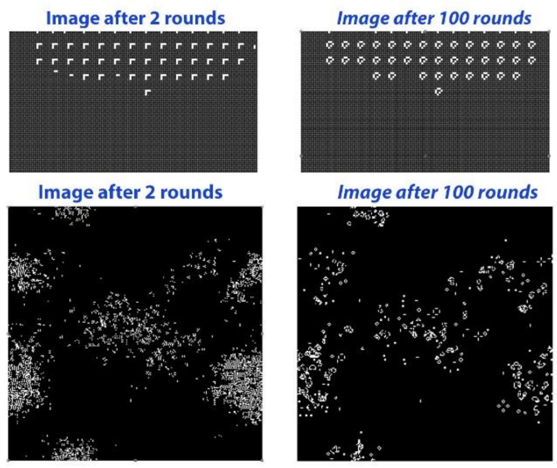


Figure 7: 256x256 image