Concurrent Computing Report on Conway's Game Of Life

Team Number : 7
Team Members: Sunny Miglani (sm15504@my.bristol.ac.uk)
Student number : 1543607

Functionality and Design :

General Structure:

The design of the program is using a single Distributor along with 4 Workers. The design is split in such a way that the Distributor first stores the "map" (the image data) into an array then splits it into 4 parts and sends each of these parts to the Workers.

The map is split horizontally into 4 equal sized row sets (each with IMHT/4 rows), allowing easy calculation for the Workers. Each Worker gets 2 extra rows on each side from the Distributor (in the first iteration) to allow them to work on their "maps" without wasting time communicating with each other.

Throughout the program the only communication between the Workers is to update the extra rows called the "buffer rows". These are updated after each iteration of the game. These buffer rows are initial given to the Workers by the Distributor and then updated by communicating amongst themselves.

Details of memory:

For efficient use of limited memory available on the xMOS board, the data in each cell is represented by a bit. Since the cells can only have two states (Dead or Alive), it makes it easy to encode large amounts of data into simple structures. For this project I've decided to use an integer array (each integer in the array stores 32 bits i.e. 32 cells). This allows for the program to handle 512x512 sized images and possibly 1024x1024 sized images.

For readability and easy coding we represent the map as a 2D array which is stored as a 1D array. The 1D array allows proper memory management and the 2D part of it allows us to easily traverse through the "map" thinking it was a 2D figure (i.e. a matrix)

Overall Timeline of Program: Distributor, Workers and helper functions (I/O timers etc) all run at the same time in a "par block" in the main method. This allows the Workers to do any initialisation required (creating arrays as needed etc).

- Distributor: Gets the values from the DataIn function bit by bit and stores it in a 1D integer array. Then splits up the data and transfers it to the Workers along with an extra "buffer row" on either side. The Distributor has constant communication with the Workers in the form of a "workOn" value mentioned in the code which allows for implementation of PAUSE and STOP parts of the program. At the end of X number of iterations, the Workers are sent a STOP signal initialising the transfer of data from the Workers to the Distributor. This data is then used to print out the image using DataOut.
- Workers: The Workers work relatively sequentially and independant of each other until the end of an iteration over the map. Each Worker has 2 maps, a

"new" map and a "work" map. The "work" map is used as a reference while any changes are done to the "new" map. These maps are updated and refreshed after every iteration. During each iteration of the map, the workers look through each cell and its neighbours and decide whether the cell should live or die based on the rules of Conway's Game Of Life. It recieves a "workOn" value from the Distributor letting it know the number of times it has to iterate over the map. At the end of each iteration it communicates with the other workers to update the "buffer rows" for the next iteration. At the end of the iterations, after recieveing th STOP signal from the Distributor the workers send their data ready to be printed.

There are multiple parts of the program which use various macros and functions to allow readability and easy debugging. It also allows for relatively cleaner code throughout the program.

Most of the Macros are defined with comments in the code itself, and anything taken from the internet is referenced at the top of the program.

Alternate Design Choices: There were multiple design choices when constructing this program, Following mentioned are a few of the options which could be explored given enough time.

- Splitting the map vertically (no expected change in speed, but would be interesting to explore)
- Using Asynchronous channels instead of current channels. (Expected a huge increase in speed after altering code to a level, especially during updating of the buffer cells)
- Using different data type to encode to bits (i.e Long, Double etc) which could create somewhat of a higher speed of communication or at least require less memory storage for the "maps"
- Using multiple distributors and removing communication b/w Workers to allow more concurrent communication between Workers.
- Using different number of Workers (2 or 3)

Testing of the System :

- Speed of 16x16, 256x256,512x512,1024x1024 after 100 rounds each (of computation ignoring I/O)
- The highest size of image is : 2^32 x (INT_SIZE_ARRAY_MAX/2)

Other Implementations of the Program:

Attempts were made to use the buttons and tilting of the board along with the LEDs to allow better user interaction. But due to a bug that cannot be found/fixed in the given timeframe, these interactions cannot happen on any image bigger than a 16x16 image. After spending multiple hours on figuring out why this happens and unable to get an answer a design decision was made that it's easier to let the code run for a predefined set of iterations than use buttons / LEDs which create bigger hassles in the code.

Along with this report and the main code (without buttons/LEDs), the code of buttons/LEDs has also been included for review if applicable.

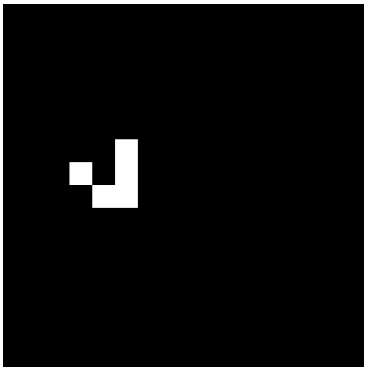
Critical Analysis of the Code/Project:

Very obvious data/time waste by allowing the distributor to input the map into an array and then distributing it to the Workers as opposed to sending the data to the Workers directly. The latter method would allow much faster I/O which while not affecting computation time would affect how fast the overall program runs and would possibly allow more memory to be used for bigger images.

Not efficient implementation of cell neighbour checking at code level (inefficient code or just hard to read). This is caused by time restraints and memory restraints. Memory is limited so creating a separate function that copied the data would be a very obvious waste of memory but would make the code much more readable and easier to debug.

Outputs and Times

Output for 16 x 16 after 2 iterations as requested :



Sizes and Timing of each image at 100 iterations

16x16

Time : 4991258 ===> 0.04991258 seconds.

64x64

Time : 73421038 ===> 0.73421038 seconds.

256x256

Time :1106763638 ===> 11.06763638 seconds.

512x512

Time : 247807852 with 2 overflows in int of size "2,147,483,647"

Therefore approx total time : (247807852 + 2(2,147,483,647)) / 10^8

= 45.42775146 seconds

1024x1024

Time : 288829187 with 6 overflows in int of size "2,147,483,647"

Therefore approx total time : (288829187 + 6(2,147,483,647))/ 10^8

= 131.73731069 or 2.19 minutes.