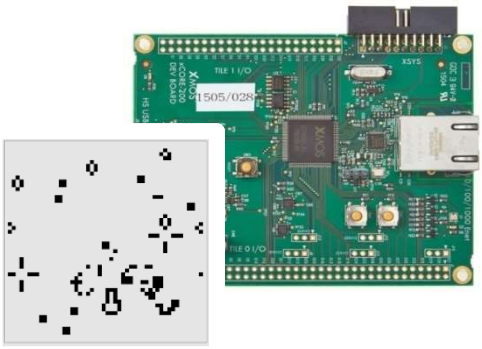
COMS20001 Concurrent Computing (2018-19)

Assignment Weeks 5-10

(pair programming work, summative, 25%)

xCore-200 Cellular Automaton Farm



Overview: This assignment is the 1st summative coursework for COMS20001. It runs over approx. 5 weeks. The coursework is worth 25% of the unit mark. It is to be completed in programming teams, which are registered on SAFE. You must report any change to pairings to the unit director BEFORE starting your assignment. Meet regularly and make sure you manage your team well. Let us know about issues before they grow to affect your team's performance.

Submission: Every student is required to upload their full piece of work (incl all XC source and project files, and your PDF report) as a single ZIP file to SAFE before Thu 6th December 2018, 17:59. Make sure you submit it early (not last minute!) to avoid upload problems. Each team member has to upload an identical copy of the zip file containing all of the team's work.

Assessment: You will be marked on your code, system design, experiments and report, and your understanding of it. Your team is assigned a presentation time slot published on the unit website. Presentation sessions will be held over the last 3 weeks of term at MVB2.11. Times will be published on the unit website. Both team members must be present to get a mark. During the presentation we will run and discuss the submitted program (using xCore-200 kits for a live demo and discussion). We will ask you questions about your work and you will be able to showcase the merits of your project. Do not plagiarise. Both team members should understand all code developed in detail.

Your Task

Introduction. The British mathematician John Horton Conway devised a cellular automaton named 'The Game of Life'. The game resides on a 2-valued 2D matrix, i.e. a binary image, where the matrix entries (call them cells, picture elements or pixels) can either be 'alive' (value white 255) or 'dead' (value black 0). The game evolution is determined by its initial state and requires no further input. Every cell interacts with its eight neighbour pixels, that is cells that are horizontally, vertically, or diagonally adjacent. At each matrix update in time the following transitions may occur to create the next evolution of the domain:

John Horton Conway
(Photo by Thane Plambeck)

- any live cell with fewer than two live neighbours dies
- any live cell with two or three live neighbours is unaffected
- any live cell with more than three live neighbours dies
- any dead cell with exactly three live neighbours becomes alive

Consider the image to be on a closed domain (pixels on the top row are connected to pixels at the bottom row, pixels on the right are connected to pixels on the left and vice versa). A user can only interact with the Game of Life by creating an initial configuration and observing how it evolves. Note that evolving such complex, deterministic systems is an important application of scientific computing, often making use of parallel architectures and concurrent programs running on large computing farms.

Task Overview. Your task is to design and implement a concurrent multi-threaded program on the xCore-200 Explorer board, which simulates the 'Game of Life' on an image matrix. The board's buttons, orientation sensors, and LEDs should be used to control and visualise aspects of the game. The game matrix should be initialised from a PGM image file and the user should be able to export the game matrix as PGM image files. Your solution should make efficient and effective use of the available parallel hardware of the xCore-200 architecture by implementing farming or geometric parallelism, and communication of parts of the game matrix across several cores/tiles based on message passing.

Skeleton Code. To help you along, you are given a simple sample project, which showcases some of the routines needed for the task (e.g. reading/writing image files, using the orientation sensors). You should make a copy of the whole directory structure provided in your workspace and import the entire workspace COMS20001. Clean all projects and rebuild the project game_of_life. After starting the program on the xCore-200 Explorer kit the system waits for your board to be physically tilted, then reads in an image (from a PGM image file) using a thread DataInStream that produces a stream of pixel values (describing the image matrix left to right, line by line) sent on an XC channel to a distributor thread that simply inverts the image. The program writes the resulting stream of pixel values to an image (PGM image file) using a thread DataOutStream. The provided example code operates on example images of size 16x16 pixels only. Feel free to change any of the skeleton code provided or add skeleton code from any of your previous assignments. Your finished system should evolve the "Game of Life" forever; you do not need to shut down your system gracefully.

Process Control and User Feedback. In your application use buttons and the orientation sensors on the xCore200 Explorer board for control and LEDs for state visualisation:

• *Button SW1:* This button should start the reading and processing of an image, indicate reading by lighting the green LED, indicate ongoing processing by flashing of the other, separate green LED alternating its state once per processing round over the image.

• *Button SW2:* This button should trigger the export of the current game state as a PGM image file, indicate an ongoing export by lighting the blue LED

• *Orientation Sensor:* Use the physical X-axis tilt of the board to trigger processing to be paused, and continued once the board is horizontal again, indicate a pausing state by lighting the red LED, print a status report when pausing starts to the console containing the number of rounds processed so far, the current number of live cells and the processing time elapsed after finishing image read-in.

Building a Process Farm. To perform the evolution of the game, your program should implement a 'distributor' or 'farmer thread' that tasks different 'worker threads' to operate on different parts of the image matrix in parallel, which should be stored across the local memory of different cores to allow for larger images. Consider packing your images (represent each cell with one bit rather than a byte) to be able to store/process images more space-efficiently.

Your Report. You need to submit a CONCISE (strictly max 4 pages) report which should contain the following sections:

Functionality and Design (1 page max): Outline what functionality you have implemented, which problems you have solved with your implementation and how your program is designed to solve the problems efficiently and effectively.

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.

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.)

Workload and Time Management.

It is important to carfully manage your time for this assignment. You should not spend much over 25 hours on this task, this is about 5h per week over 5 weeks incl. labs. Do not spend hours trying to debug on your own; use pair programming, seek help from our teaching assistants during scheduled labs, use the forum, or see a lecturer.

Further Task Details and Assessment Guideline.

The markers will take into account your lab presentation, your report and your source code. Find below a guideline for assessment. However, this is only a guideline and submissions will be marked on an individual basis considering your understanding of your system, as well as the quality, quantity and presentation of work conducted.

To pass the assignment you should at least implement a working, concurrent system for the xCore-200 Explorer board that correctly evolves an image according to the rules of Game-of-Life and submit a report that discusses your implementation to a basic standard.

For a mark around 50 make sure you submit a clean, well documented piece of code that uses multiple worker threads effectively for evolving the Game-of-Life for a few rounds.

For a mark around 55 also make sure that your system is well-tested (e.g. deadlock-free), and your report is concise and discusses key aspects of your system clearly. Make sure you implement the correct button, board orientation and LED behaviour.

For a mark around 60 implement a system that can process larger images (above 512x512 pixels) using memory on both tiles; and show a good understanding of the concurrency concepts relevant and/or implemented.

For a mark of merit around 65 use timers to measure the throughput/processing speed of your system when evolving your game over 100 consecutive processing rounds (excluding I/O) and on different inputs (e.g. small vs. large images, different initial conditions etc). Discuss your results systematically and concisely.

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.

Higher first class marks (70+) are reserved for excellence beyond the above. Here we look for work that implements a well tested system, which also allows for, for instance, dynamically adapting the strategy of storing image parts or distributing tasks to workers based on the game state, or dynamically taken performance measurements, or your own ideas how this system can be turned into something more efficient, flexible and/or sophisticated … feel free to ask for the possibility of modifying the physical board etc to implement your own ideas. Note that for top marks you should show an excellent understanding of the concurrency concepts in and beyond the task.

Keep in mind that together with the marking indications outlined above, your understanding of your own code during the lab presentation, as well as at least minimal code readability will be considered for assessment. □