Report (Prog06)

2020.11.23

Baheem Ferrell

1. The Cellular Automation

1) Version 1 : Multithreaded CA with short-lived threads

At each rendering step, the main thread performs below actions repeatedly.

- *creates numLiveThreads* separate threads
- assigns non-overlapping portions of 2D grid to each thread
- waits until all these threads are finished
- destroys all child threads
- updates the 2D grid by swapping current and next arrays
- renders the grid on the screen
- registers the next rendering callback to itself.

ThreadInfo struct is modified to include *lowerBound* and *upperBound* which together indicate the range of rows that one thread is assigned to calculate. An instance of this struct is passed when a new thread is created. When registering the next rendering callback, *renderDelay* is used to control rendering speed so that it can be modified using + and - keystrokes. ***Note. In order to conform to the specification of script2.sh, Version 1 requires an integer through standard input at the beginning of execution. It blocks until an integer is read. This integer is used to set the window title as well as to locate the corresponding named pipe.***

2) Version 2 : Multithreaded CA with a single mutex lock

In order to implement Version 2, a barrier lock is used.

A mutex lock is efficient in not allowing a critical section to be executed by more than 1 thread at any time. But in this project, each calculation thread must calculate the next step only once before the main thread collates the results and renders them. This behavior closely resembles that of a barrier lock.

A barrier lock is initialized with the number of threads equal to *numLiveThreads* + 1, because there are *numLiveThreads* separate calculation threads in addition to the main thread. At each rendering step, *pauseThreads* is set to prevent all calculation threads from accessing the grid after calculation is done and the barrier lock is obtained to synchronize all calculation threads at the location where the calculation is finished. Then the main thread collates and renders the result, and then clear *pauseThreads* so that all calculation threads resume calculating next step.

3) **Version 3 : Randomized multithreaded CA with a grid of mutex locks**

A lock is created per each cell in the 2D grid.

Each calculation thread randomly selects a valid cell from the grid, acquires 3x3 locks around the cell, calculates the next evolution for the cell and releases the locks.

At each rendering step, all locks are acquired to prevent any calculation thread accessing the grid before rendering the grid. After the rendering all locks are released immediately.

Performance test on this initial implementation was found to be very inefficient because each thread was racing each other to acquire maximum resources. Adding *usleep(5000)* call immediately after releasing 3x3 locks was found to be an efficient solution to this problem, as it allows other threads to acquire necessary locks and perform calculation.

4) **Named pipes : Additional feature for script2.sh**

In order to conform to the specifications of script2.sh, a named pipe is used as a non-blocking input for external commands for all versions.

At the beginning of execution, the application expects an integer through standard input. This integer value is used to set the *glut* window title as to locate the pipe. ***The execution is blocked until this integer is given through the standard input.***

A separate thread is created to handle reading external commands from the given pipe. It is implemented as a non-blocking read in order to ensure that the thread responds to *cleanupAndquit*() function call of the main thread.

2. **Scripts**

1) **script01.sh : Single launcher**

This script builds CA and executes it with the given arguments.

It expects 3 input arguments, i.e., number of rows, number of columns and number of threads.

Then it launches CA with the given arguments.

When a command is typed in through standard input, it is written into a named pipe from which the application reads the command and executes it.

2) **script02.sh : Command parser**

This script does not expect any input arguments.

It is assumed that CA is executed with the following command.

cell numOfRows numOfCols numOfThreads

In the script, command is read from the standard input line by line.

A line is split into an array at white spaces. (Before splitting ":" is replaced by " "(white space).)

If the first word is “cell”, a counter variable is incremented by 1 and the remaining words are used as arguments to execute a CA with the current counter as the process index.
If the first word is an integer, the remaining words are concatenated and written to a named pipe to be read by corresponding CA.

3. Miscellaneous

In this project, two assumptions are made regarding the input of process index and the number of threads.
Each CA executable expects an integer through the standard input before start entering the main loop. (Refer to Section 1.1 and 1.4)
Instead of cell width height as in the project description 4.3, *cell numOfRows numOfCols numOfThreads* is used. (Refer to Section 2.2)