CSC-213: PROF. CURTSINGER
FINAL PROJECT REPORT

ANNA BLINDERMAN, DAVID KRAEMER, ZACHARY SEGALL

Project Overview (<400 words)

- introduce Game of Life
- describe GUI
- describe GPU stencil update
- describe listener CPU threads
- describe evaluation strategies
- summarize evaluation results

We implemented a variation of Conway's Game of Life – a cellular automaton simulation created by the mathematician John Conway in order to explored the role of various types of optimization within it. Scientific computing and large-scale simulations are incredibly useful across a variety of fields. However, many of these programs are prohibitively slow. Therefore the algorithms and design of these programs is crucial.

[TRANSITION] Life is simply a series of iterations of a rendering of a grid of cells, all of which are either dead or alive. Each cell's state is determined by the states of its direct neighbors. Life provides an interesting basic model for population growth and is an ideal demonstration of complex systems arising from simple rules.

[davis can you please write the paragraph on the ui don't you read through libraries in your spare time or something]

For each iteration, we use Conway's algorithm to determine the state (dead or alive) of every cell in our grid. Because this algorithm takes as input a cell along with the states of each of its eight immediate neighbors, this algorithm naturally lends itself to an "embarrasingly parallel" stencil computation on the GPU.

We also have two threads running on the CPU – one for recording and acting on user input from the mouse; the other from the keyboard.

To evaluate our system, we first varied the number of threads per block in our stencil function calls. Then, we implemented a feature that allowed us to find regions with no live cells in order to skip their calls to the update functions.

[LOL SUMMARY OF RESULTS WHAT HAPPENED?!?!?]

Design and Implementation ($\sim$2 pages)

- summary of overall implementation structure
- Component 1: GUI
    - responsibilities/how fits into entire structure
    - rationale for choice
    - data structures/algorithms/libraries/other details
- Component 2: mouse/keyboard input
    - responsibilities/how fits into entire structure
    - rationale for choice
    - data structures/algorithms/libraries/other details
- Component 3: GPU stencil update
    - responsibilities/how fits into entire structure
    - rationale for choice
    - data structures/algorithms/libraries/other details
- Butler Lampson's Hints for Computer System Design (integrate?)
    - don't reinvent the wheel (+ how it did/didn't help)
    - be prepared to throw an entire thing out (+ how it did/didn't help)
- figures if appropriate

The rules to Conway's Game of Life are simple. There exists a grid of cells, where every cell is either dead or alive. At each iteration of the simulation, Conway's algorithm is applied to every cell in order to determine its next state. The algorithm is defined as follows:

- Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with three live neighbours becomes a live cell, as if by reproduction.

[CITE WIKIPEDIA, ALSO INSERT PICTURE] adjustible things:

- GUI (game board) display size
- cell size (thus adjusting the number of cells)
- delay between iterations while running simulation
- colors, including linear interpolation between start and end values
- can randomize a board
- can load board from file

During the simulation, the user also has the ability to:

- left-click (or drag) to bring dead cells to life
- right-click (or drag) to kill live cells
- press `Ctrl-Q` to quit the simulation (close the GUI)
- press `Ctrl-C` to clear the board (set all cells to dead)
- press `Ctrl-P` to pause the simulation (set the global `running` variable to false)
- press `Ctrl-Space` to advance the iteration one step (run the update function just once)
- press `Ctrl-G` to add a "glider" shape to the board [DAVIS HOW U DO DIS]

Given that the update algorithm is itself simple to implement, most of our difficulties were centered around coordinating user input with the GUI display and the update algorithm.

Initially, we intended to use a scheduler (as in the Worm lab) in order to split up listening for input and updating across processes. We quickly ran into problems with this approach, as the listeners would sometimes fail to register input if they weren't being executed by the schedule at the exact moment the user clicked or pressed a button. [SAY THIS IS ACTUALLY LAMPSON MAYBE] Thus, we realized we needed complete concurrency on the CPU and switched to an implementation with threads: one for mouse input, one for keyboard input, and the main one to run updates.

[DAVIS PLEASE TALK ABOUT SDL CAUSE LOL WHAT IS THAT] [AND ALSO BITMAP THX UR THE BEST]

In order for the threads to properly take in and act on user input, we created a struct

to contain the information necessary for the various actions in the program to be taken – the location of the cell from which the user input was recorded, the state of the mouse (clicked or unclicked), and the SDL_event which indicates [IS THE THE KEY DAVIS PLS HELP HELP PLS]. The mouse thread is then responsible for executing the first two of the above bullet points when appropriate; the keyboard thread is responsible for the other five. The main thread calls our update function. In any case, this struct of arguments is passed to the functions so that they may act appropriately.

For the simulation itself to run, we must update all cells in our board according to Conway's original algorithm. As described above, this algorithm naturally lends itself to an "embarrassingly parallel" implementation. Specifically, we used a stencil pattern since a cell's next state depends on the states of its eight immediate neighbors and we can process whether or not a cell survives independent of this determination for any other cell on this board. Each time the update function is called, we first copy over our

a call to both of two GPU functions is made.

It takes place on a grid, so we need the GUIRegarding the more general mechanics of our project, we will use an SDL_surface to create a board as in the Galaxy lab. The cells on the board will be initially populated from either a random starting configuration, one of some set of preset starting configurations, or by the user clicking squares on the board. We will have a struct to represent the state of the board and will use a bitmap for the visualization of the board. Because our internal representation will be separate from our visualization, we will need to translate from the internal representation (the struct) to the visualization (the bitmap) at each interval of time. Further, this separation will allow us to more easily manipulate and update the representation and then simply update the visualization afterwards using Conway's original algorithm.

Evaluation

- describe experimental setup
    - hardware
    - software (libraries)
    - data-gathering methods
- explain what trying to measure (speedup)
    - vary block size
    - implement thing that only updates regions with living things?
    - implement serial version?
- include at least one graph with at least 8 data points
- discuss interpretation of results