CEG 4136 Computer Architecture III
Parallella Project Report

Conway's Game of Life
GROUP #16

Professor:	Miodrag Bolic	
Group:	16	
Students:	Conrad Berlinguette	#7005055
	Alexandre Plante	#7908179
Date:	03 December 2018	

OBJECTIVES

Implement and demonstrate Conway's game of life on the Parallella board, and roughly calculate the speedup time. The project used the code repository: *https://github.com/parallella/parallella-examples/tree/master/game-of-life*

EQUIPMENT

- Parallella board
- Ethernet cable
- MicroSD card
- PuTTY software
- Notepad++

OVERVIEW

In 1970, British mathematician John Conway devised a game concerning cellular automaton. The game is a discrete model studied in computer science modeling, where an *n* by *n* grid of cells is supplied. Each cell is binary: 'x' (dead) or 'o' (alive). For each cell, a decision is made mathematically on whether it lives, dies, or multiples.

The algorithm can run for an arbitrary number of iterations, and each new iteration is calculated from the previous one. Each value is calculated only by using its neighbour's values, and it is therefore very parallelizable. Depending on the initial conditions, the cells form various patterns throughout the course of the game. The rules are follows:

- For a space that is 'populated':
    - Each cell with one or no neighbors dies, as if by solitude.
    - Each cell with four or more neighbors dies, as if by overpopulation.
    - Each cell with two or three neighbors survives.
- For a space that is 'empty' or 'unpopulated':
    - Each cell with three neighbors becomes populated.

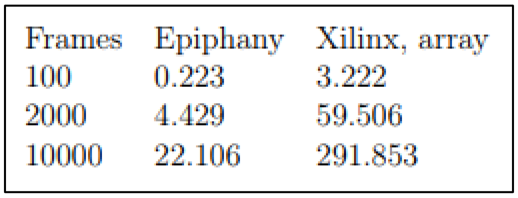
GAME OF LIFE

Noémien Kocher wrote code for a variation of the game to run on the Parallella in 2016. He set a fixed grid size of 4x4, which equates to 16 cells, meaning one per processor. The user is permitted to set the number of iterations/state changes they wish to see. We added a timer feature to calculate the time spent between states, to give us a better understanding of how fast the calculations were performed.

On average, each iteration took approximately 0.2 milliseconds. Most of that processing time would have been compounded by the number of cells if run in a serial fashion, although we lacked the control single out only one core for processing. Because we do not have a precise baseline, our speedup time is more of an assumption.

The 0.2 milliseconds represents the time it takes for a Next State to be generated from a Previous State. We know that a cell's Next State is independent of any other cell's Next State, and therefore its Next State calculation can be parallelized. In our case, each cell can be independently calculated by its own dedicated processor. This means there is initialization overhead, and likely some minor iteration or printing overhead, but the bulk of the program (i.e. calculating the cell state at each iteration) is completely parallelized. This would lead us to an assumption that the speedup of a 4x4 grid, from a single core to 16 cores, would be close to a value of 16. This result should stand more or less regardless of the number of iterations, the greater the number, the closer to a speedup of 16.

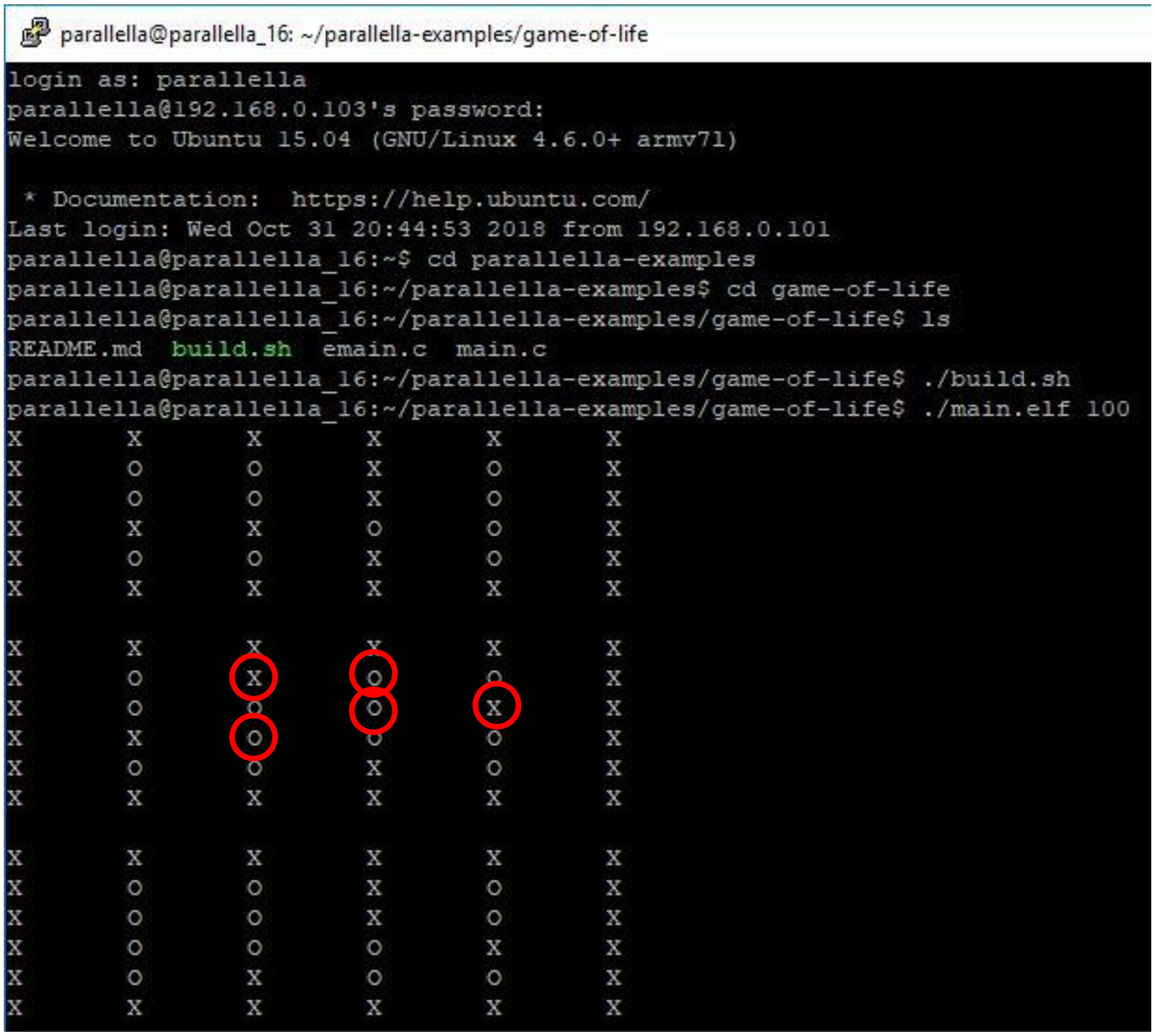
We also did some research on the game and what other people had done as implementations. It appears our estimations were not far off, as Trygve Aaberge (https://daim.idi.ntnu.no/masteroppgaver/011/11745/masteroppgave.pdf) wrote a report and found that the Epiphany was 13 to 16 times faster than the main single core processor. In his case, the grid size was significantly larger (180x180), and his results for total program runtime, in seconds, were as follows:



Frames	Epiphany	Xilinx, array
100	0.223	3.222
2000	4.429	59.506
10000	22.106	291.853

CONCLUSION

Conway's Game of Life provided a fun opportunity to visually represent parallel processing. Because of the inherit independence of the game's elements, we could easily implement parallelization for the bulk of the program. Our findings proved similar to other field research, and we are comfortable in our implementation and assumptions to say that the speedup offered by the Parallella board is close to 16.


parallella@parallella_16: ~/parallella-examples/game-of-life

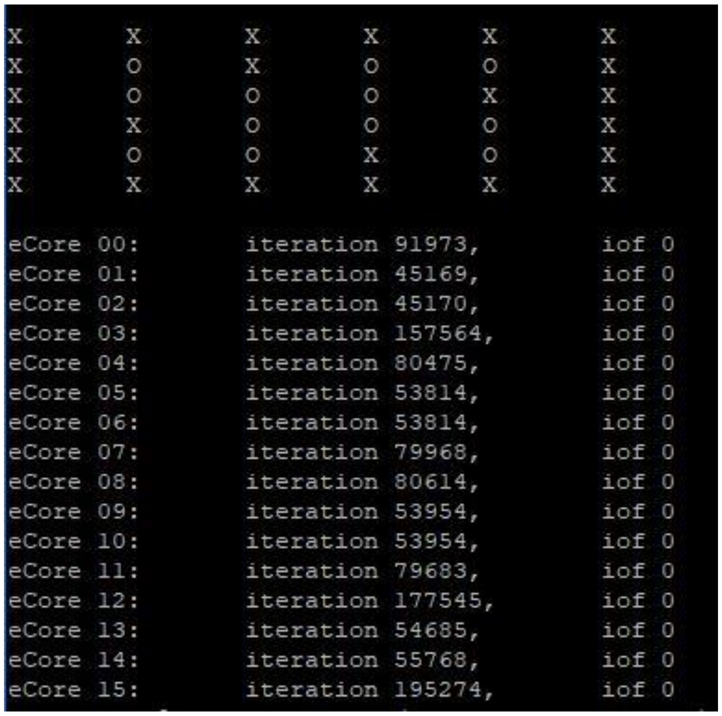
```
login as: parallella
parallella@192.168.0.103's password:
Welcome to Ubuntu 15.04 (GNU/Linux 4.6.0+ armv7l)

 * Documentation:  https://help.ubuntu.com/
Last login: Wed Oct 31 20:44:53 2018 from 192.168.0.101
parallella@parallella_16:~$ cd parallella-examples
parallella@parallella_16:~/parallella-examples$ cd game-of-life
parallella@parallella_16:~/parallella-examples/game-of-life$ ls
README.md  build.sh  emain.c  main.c
parallella@parallella_16:~/parallella-examples/game-of-life$ ./build.sh
parallella@parallella_16:~/parallella-examples/game-of-life$ ./main.elf 100
X       X       X       X       X       X
X       O       O       X       O       X
X       O       O       X       O       X
X       X       X       O       O       X
X       O       O       X       O       X
X       X       X       X       X       X

X       X       X       X       X       X
X       O       X       O       O       X
X       O       O       O       X       X
X       X       O       O       O       X
X       O       O       X       O       X
X       X       X       X       X       X

X       X       X       X       X       X
X       O       O       X       O       X
X       O       O       X       O       X
X       O       O       O       X       X
X       O       X       O       O       X
X       X       X       X       X       X
```

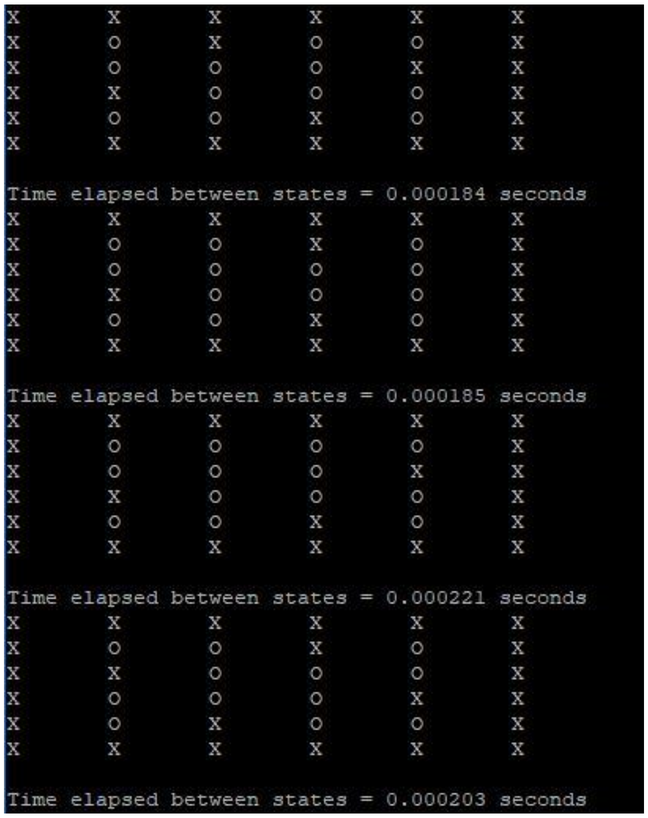
Figure 1: *Demonstration of a State Change*



```
X       X       X       X       X       X
X       O       X       O       O       X
X       O       O       O       X       X
X       X       O       O       O       X
X       O       O       X       O       X
X       X       X       X       X       X

eCore 00:       iteration 91973,        iof 0
eCore 01:       iteration 45169,        iof 0
eCore 02:       iteration 45170,        iof 0
eCore 03:       iteration 157564,       iof 0
eCore 04:       iteration 80475,        iof 0
eCore 05:       iteration 53814,        iof 0
eCore 06:       iteration 53814,        iof 0
eCore 07:       iteration 79968,        iof 0
eCore 08:       iteration 80614,        iof 0
eCore 09:       iteration 53954,        iof 0
eCore 10:       iteration 53954,        iof 0
eCore 11:       iteration 79683,        iof 0
eCore 12:       iteration 177545,       iof 0
eCore 13:       iteration 54685,        iof 0
eCore 14:       iteration 55768,        iof 0
eCore 15:       iteration 195274,       iof 0
```

Figure 2: *Iterations per Core after 100 State Changes*



```
X       X       X       X       X       X
X       O       X       O       O       X
X       O       O       O       X       X
X       X       O       O       O       X
X       O       O       X       O       X
X       X       X       X       X       X

Time elapsed between states = 0.000184 seconds
X       X       X       X       X       X
X       O       O       X       O       X
X       O       O       O       O       X
X       X       O       O       O       X
X       O       O       X       O       X
X       X       X       X       X       X

Time elapsed between states = 0.000185 seconds
X       X       X       X       X       X
X       O       O       O       O       X
X       O       O       O       X       X
X       X       O       O       O       X
X       O       O       X       O       X
X       X       X       X       X       X

Time elapsed between states = 0.000221 seconds
X       X       X       X       X       X
X       O       O       X       O       X
X       X       O       O       O       X
X       O       O       O       X       X
X       O       X       O       O       X
X       X       X       X       X       X

Time elapsed between states = 0.000203 seconds
```

Figure 3: *New Feature - Timing of State Changes*