

The Game of Life

Concurrent Computing Coursework

Josh Felmeden, NK18044
Antoine Ritz,

December 3, 2019

Functionality and Design

Our solution was built up by initially creating a single threaded solution to the problem. This version iterates through the board bitwise, and for each bit gathers all the 'neighbours' for the cells (the 8 directly adjacent cells). From this, the logic is applied and the cell is updated if necessary. This is repeated for the desired number of turns.

From this, we created a multi-threaded solution. We split the board up into strips and passed each strip to a worker. However, each worker would also need information from the lines directly above and below its strip of cells (called *halo lines*). We decided to pass these halo lines wrapped around the strips, so that the workers are able to calculate each cell correctly. Once they have completed their strip, they return it to the distributor function. The function reconstructs the world and begins the process again for the desired number of turns.

One problem that we ran into was that we were passing the world by means of pointer. This led to problems due to premature changes being applied to the board. To solve this, we used channels to pass the board to the workers.

The processing of the program is currently unable to be cancelled, and therefore we added the ability to quit, pause processing, and show the current state of the board with key presses. Alongside this, we also implemented an output of the number of alive cells

every two seconds using a *ticker*.

Following this, we added the ability to allow the number of workers to support all multiples of two, rather than powers of two alone. Initially, this proved difficult, since at least one worker would receive a smaller strip than the others, and consequently meant that some workers would finish sooner than others. This resulted in a **deadlock** situation, because the world reconstructing function expected all strips to be of equal size, and therefore it was blocked when the final strip was smaller than expected. To work around this, we made each worker apart from the final one work on the same number of lines, and the final worker simply had the remainder. Then, to reconstruct the world, we processed the final worker output separately from the rest.

Finally, passing the entire world between each turn is time consuming, because in reality, the only information that needs to be passed between the workers are the *halo lines*. Additionally, there is no reason to reconstruct the entire world each turn. Implementing this proved to be problematic, as sometimes the workers would get out of sync with one another. To rectify this, we implemented a *master* thread, which served as a hard limit on the speed of the other threads to ensure that the threads would not exceed one another. We passed in a new structure with `masterTurns` via pointer, which contains the current turn of the master thread.

A further problem was to implement the key presses

Tests, Experiments, and Critical Analysis

Stage 1a — Single Thread

| Benchmark | Baseline result (ns/100 turns) | Our result | % Difference |
|--------------|--------------------------------|------------|--------------|
| 128x128x2-12 | 73689886 | 2021400726 | 36% |
| 128x128x4-12 | 538915394 | 2010389496 | 26% |
| 128x128x8-12 | 261671491 | 2009465802 | 17% |

| Benchmark | Baseline CPU usage | Our CPU usage | % Difference |
|--------------|--------------------|---------------|--------------|
| 128x128x2-12 | 185% | 100% | 185% |
| 128x128x4-12 | 298% | 100% | 298% |
| 128x128x8-12 | 425% | 100% | 425% |

Average bench: 135.048s

Stage 1b — Divide and Conquer

| Benchmark | Baseline result (ns/100 turns) | Our result | % Difference |
|--------------|--------------------------------|------------|--------------|
| 128x128x2-12 | 736946012 | 1704893591 | 43% |
| 128x128x4-12 | 537751669 | 1273465510 | 42% |
| 128x128x8-12 | 362735057 | 1004595662 | 36% |

| Benchmark | Baseline CPU usage | Our CPU usage | % Difference |
|--------------|--------------------|---------------|--------------|
| 128x128x2-12 | 185% | 188% | 98% |
| 128x128x4-12 | 298% | 271% | 109% |
| 128x128x8-12 | 425% | 348% | 122% |

Average bench: 90.460s

Stage 2a — User Interaction

| Benchmark | Baseline result (ns/100 turns) | Our result | % Difference |
|--------------|--------------------------------|------------|--------------|
| 128x128x2-12 | 736121962 | 1699574829 | 43% |
| 128x128x4-12 | 537944647 | 1256970618 | 42% |
| 128x128x8-12 | 361777916 | 994380759 | 36% |

| Benchmark | Baseline CPU usage | Our CPU usage | % Difference |
|--------------|--------------------|---------------|--------------|
| 128x128x2-12 | 185% | 188% | 98% |
| 128x128x4-12 | 296% | 273% | 108% |
| 128x128x8-12 | 426% | 343% | 124% |

Average bench: 88.539%_s

Stage 2b — Periodic Events

| Benchmark | Baseline result (ns/100 turns) | Our result | % Difference |
|--------------|--------------------------------|------------|--------------|
| 128x128x2-12 | 735806211 | 1718965842 | 42% |
| 128x128x4-12 | 532913881 | 1277891763 | 41% |
| 128x128x8-12 | 361377377 | 998142692 | 36% |

| Benchmark | Baseline CPU usage | Our CPU usage | % Difference |
|--------------|--------------------|---------------|--------------|
| 128x128x2-12 | 185% | 190% | 97% |
| 128x128x4-12 | 299% | 274% | 109% |
| 128x128x8-12 | 424% | 347% | 122% |

Average bench: 94.175_s

Stage 3 — Division of Work

| Benchmark | Baseline result (ns/100 turns) | Our result | % Difference |
|--------------|--------------------------------|------------|--------------|
| 128x128x2-12 | 736951655 | 1730991762 | 42% |
| 128x128x4-12 | 537184229 | 1264782083 | 42% |
| 128x128x8-12 | 363069928 | 987918516 | 36% |

| Benchmark | Baseline CPU usage | Our CPU usage | % Difference |
|--------------|--------------------|---------------|--------------|
| 128x128x2-12 | 184% | 190% | 96% |
| 128x128x4-12 | 299% | 274% | 109% |
| 128x128x8-12 | 425% | 347% | 122% |

Average bench: 90.832_s

Stage 4 — Cooperative Problem Solving

| Benchmark | Baseline result (ns/100 turns) | Our result | % Difference |
|--------------|--------------------------------|------------|--------------|
| 128x128x2-12 | 735620822 | 716510865 | 102% |
| 128x128x4-12 | 530949831 | 483555762 | 109% |
| 128x128x8-12 | 361530673 | 333119799 | 108% |

| Benchmark | Baseline CPU usage | Our CPU usage | % Difference |
|--------------|--------------------|---------------|--------------|
| 128x128x2-12 | 185% | 289% | 64% |
| 128x128x4-12 | 299% | 425% | 70% |
| 128x128x8-12 | 425% | 667% | 63% |

Average bench: 41.679s

Conclusions

