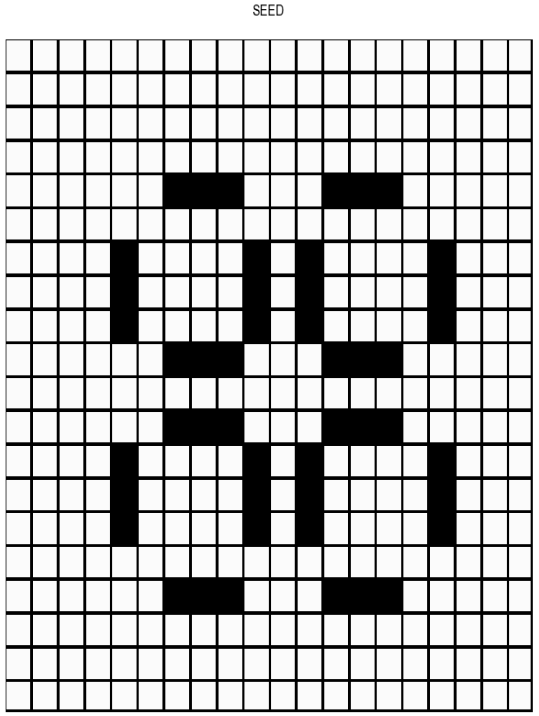
Conway's Game of Life
HPC project

Vaibhav Amit Patel
Tanmay Patel

Advisor: Dr. Bhaskar Chaudhary
DA-IICT

Conway's Game of Life

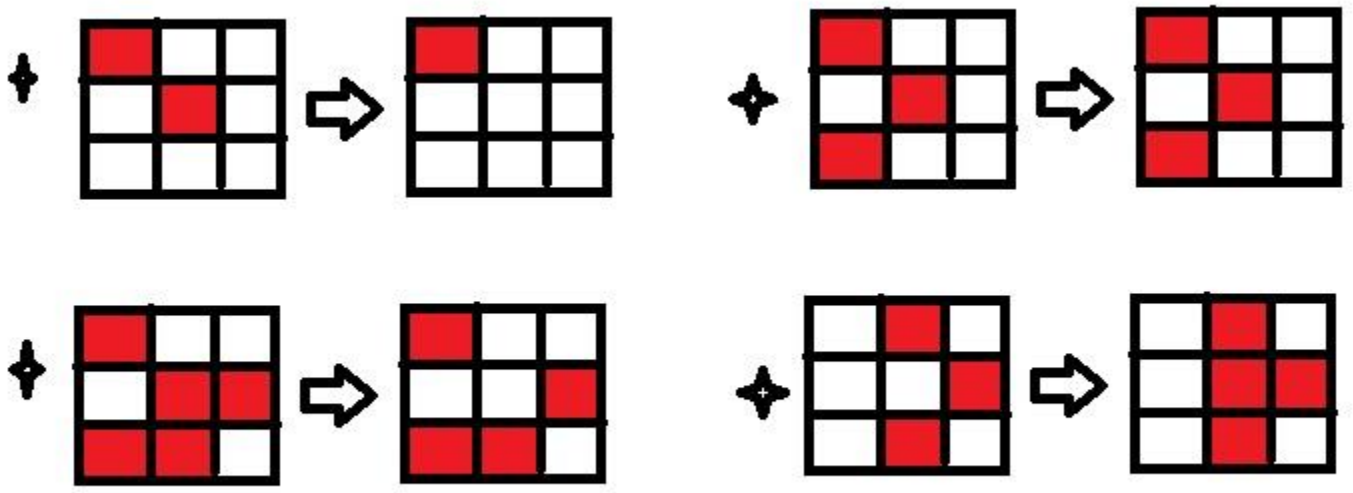
- The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by John Conway.
- Is a universal Turing machine
  - Can imitate any algorithm
- So many applications



Source: http://research.microsoft.com/mesh/

Algorithm:

For understanding the algorithm, we first need to know the rules of the game.



Time Complexity

- O((8+4) R*C)
  - R=row C=column
  - For a single step
- We can think of other alternative algorithms for this problem.
  - Prefix sum
  - Loop unroll
  - Sliding window

Speedup

- Possible speedup = P (processing elements)
- Embarrassingly parallel problem
  - The relation should be S=mP where m is close to 1 (from the left side)).
    - S=speedup
    - P=processors

Profiling

% time	Cumulative seconds	Self-seconds	Self-calls	Function name
86.32	5.21	5.21	100000000	get neighbors
13.97	6.06	0.84	10000	update
0.17	6.07	0.01	-	show

- Most of the time consumed inside the get_neighbors() function which counts number of alive neighbors for a given cell.

Parallelization strategy

- For every step we parallelize outer loop that iterates through each row. Hence, dividing rows among threads.

- Using only single grid for keeping current state of cell will require use of critical section to update edges between threads. This slows down the code because the processor will try to maintain the cache coherency at the cost of speed.

Parallelization strategy

- Hence we used temporary grid which preserves next state from current state and number of neighbors . After the next state grid is filled, we swap it with current state grid.
- This will increase speed at the cost of increase in memory space.

Optimization tricks

- Trick 1:

```
int ni = (i  + dx[k] + r)%r;
int nj = (j +  dy[k]+c)%c;
```

- Benchmarking results are (on single core):
- Old = 0.988
- New = 0.673
- 1.49 times faster.

```
int ni = i + dx[k];
int nj = j + dy[k];
if(ni < 0) ni += r;
if(nj < 0) nj += c;
if(ni >= r) ni -= r;
if(nj >= c) nj -= c;
```

Optimization tricks

- Trick 2 : In profiling there we so many calls to the get_neighbor() function so we merged the two function. Here, we are losing an important aspect of programming practice i.e. code should be Modular. This improvement will make the code difficult to read and debug.
- Speed improvement:
  - 1.05 times faster.
  - Cumulative=1.564
- Trick 3 : If conditions that we used in trick 1 to avoid modulo operator are needed only for border cells. Inner cells will not require if conditions.
- Speed improvement:
  - 1.443 times faster.
  - Cumulative=2.257

Optimization tricks

- Trick 4 : Now, each time calculating all the 8 neighbors is not optimized. So, now we are using only 3 variables v1, v2 and v3. Now in every iteration we will just calculate neighbors (just like sliding window) which will make it faster.
- Speed improvement:
  - 1.636 times faster.
  - Cumulative=3.69

1	0	1	0
0	0	1	1
1	1	1	0

V1 2	V2 1	V3 3

```
alive_neigbours = v1 + v2 + v3;

v1 = v2;

v2 = v3;

v3 = 0;

//calculate v3 again
```

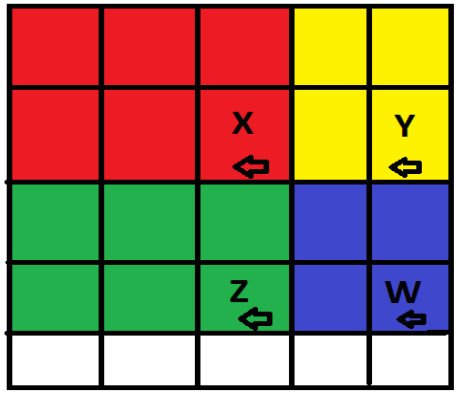
Optimization tricks

- Trick 5 : The last optimization is that we explicitly unroll and hardcoded the neighbor calculating algorithm. Before it was getting indexes from an 8 * 1 array because now it is hard coded it will be accessed from registers instead of memory or cache.
- Speed improvement:
- 1.339 times faster.

- There were so many other minor improvements were done on the serial code. **There were mainly on the idea of cache reuse. Even if we want to access a same element for 2 times we used a new variable so that second time it is used from register (because it is faster than cache).**

Another implementation

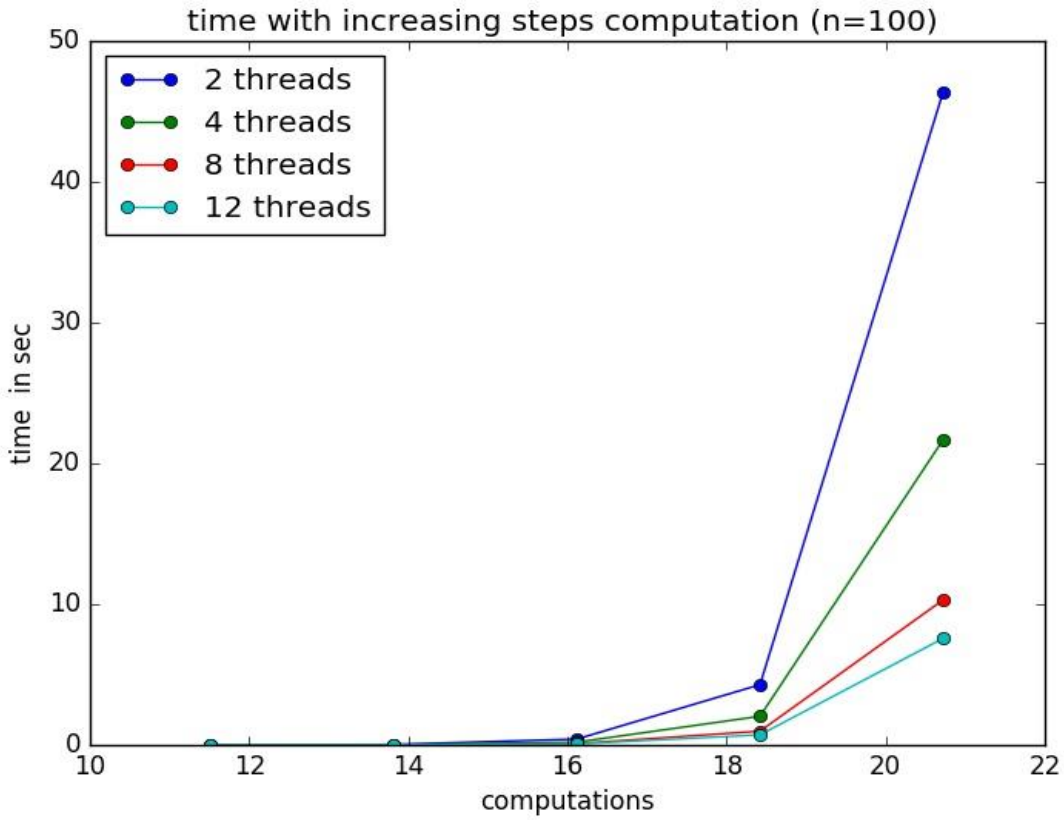
- We've implemented another algorithm for calculating neighbors and it is also 4 times faster than the serial algorithm without this lengthy code. But the other version is nearly 5 times faster that is why we are parallelizing that version.
- **And that method is prefix sum over matrix.**



Blue portion= W -Y -Z+X

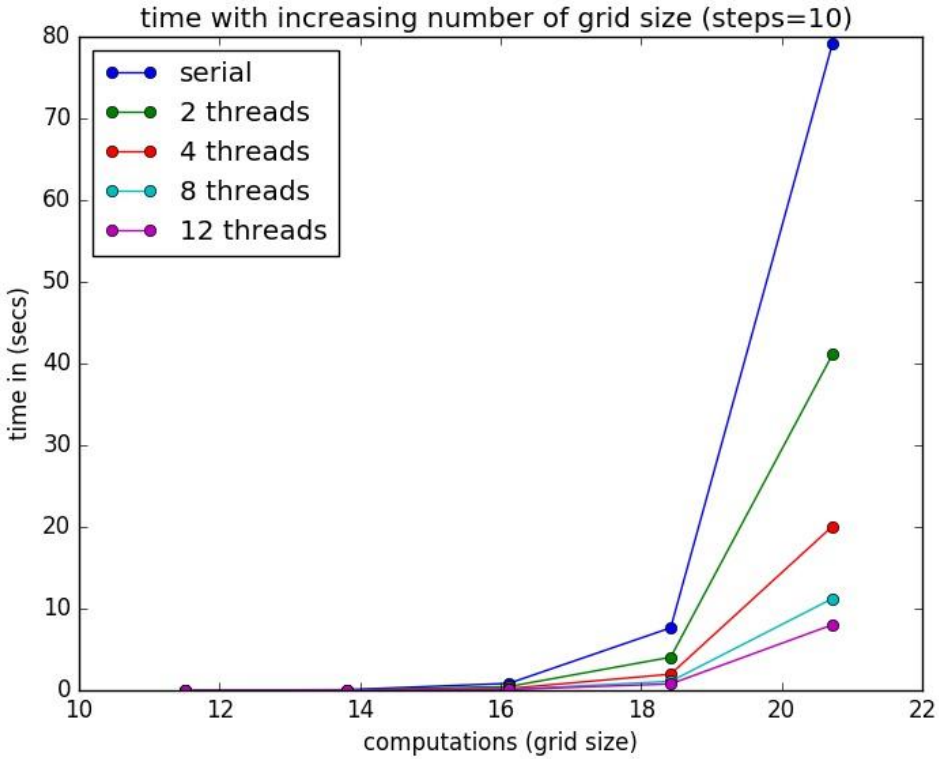
Benchmarking

Time for Increasing steps



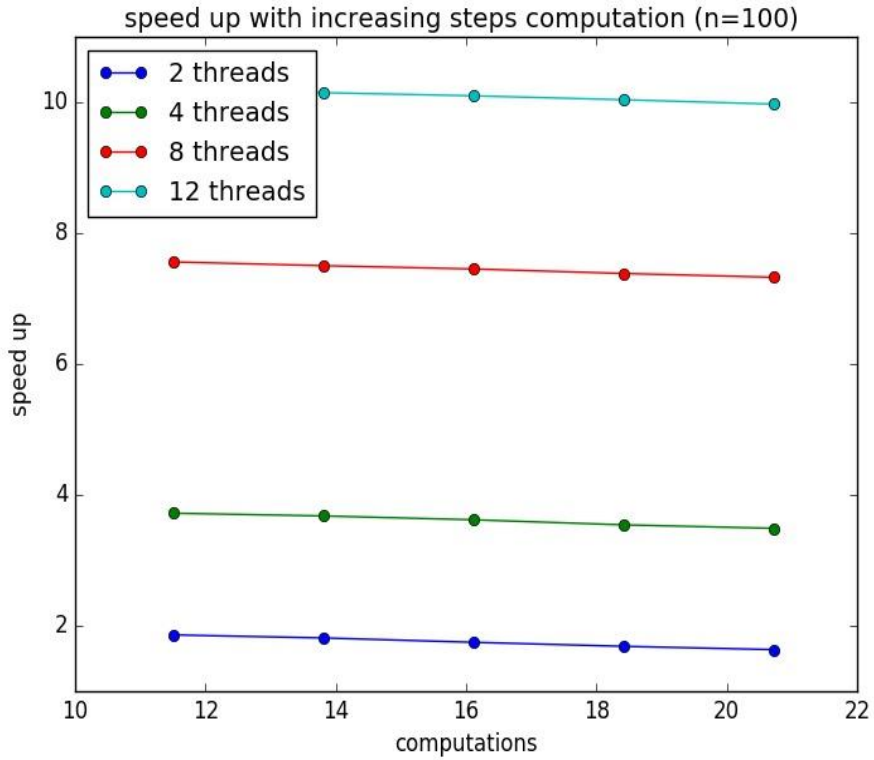
Benchmarking

Time for Increasing Grid size



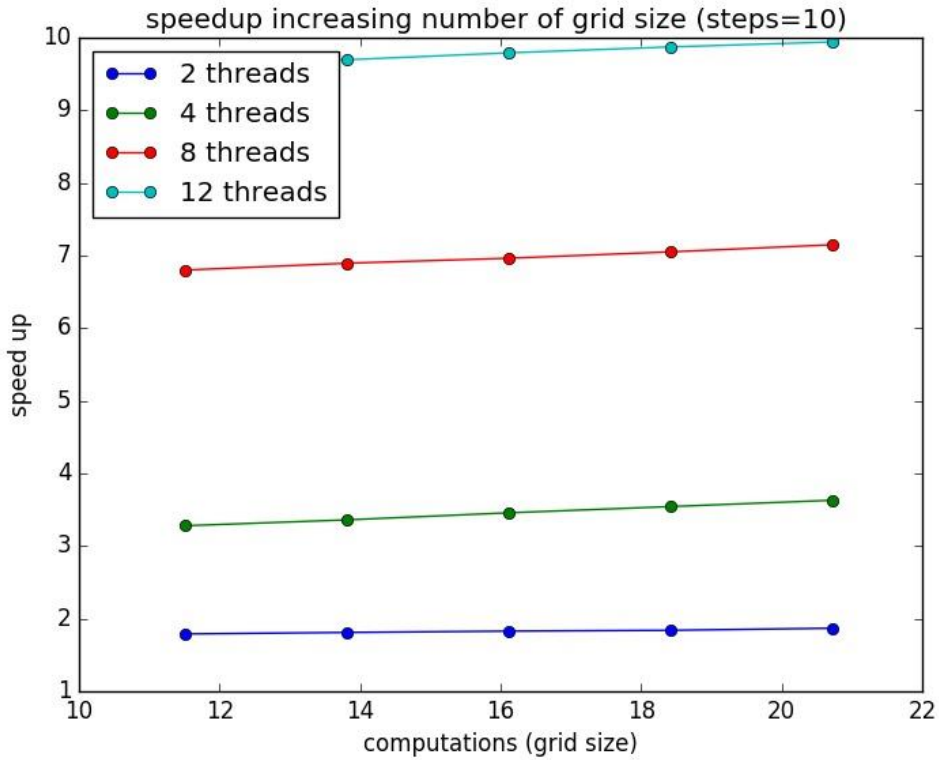
Benchmarking

Speedup for Increasing steps



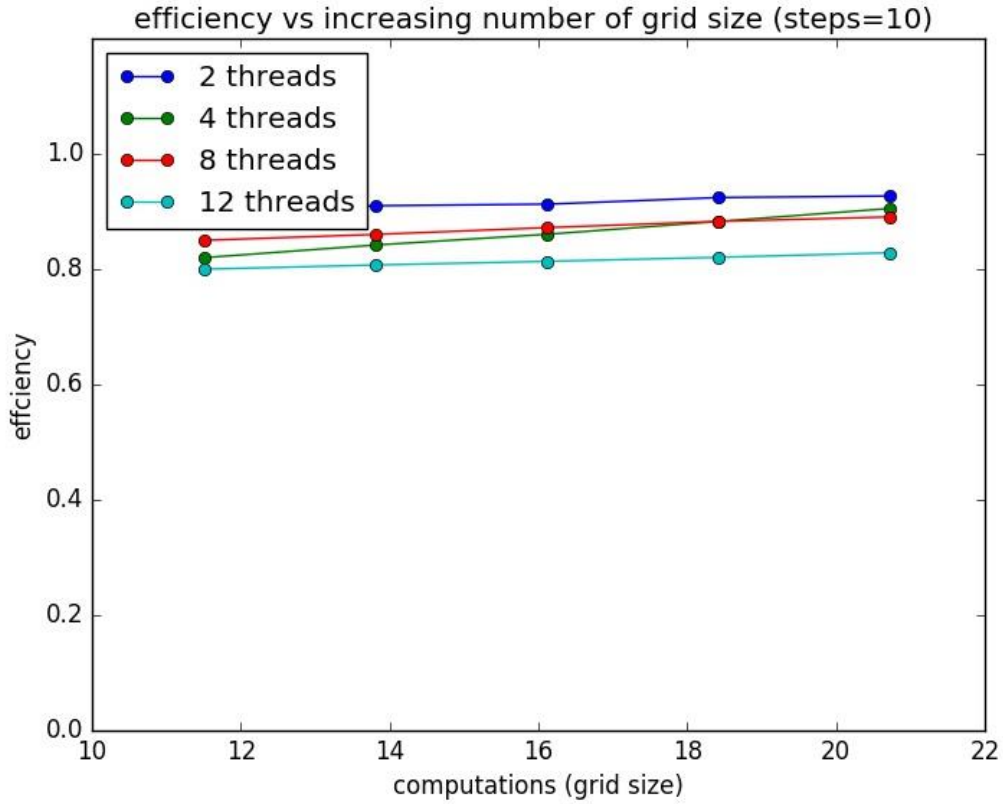
Benchmarking

Speedup for Increasing Grid size



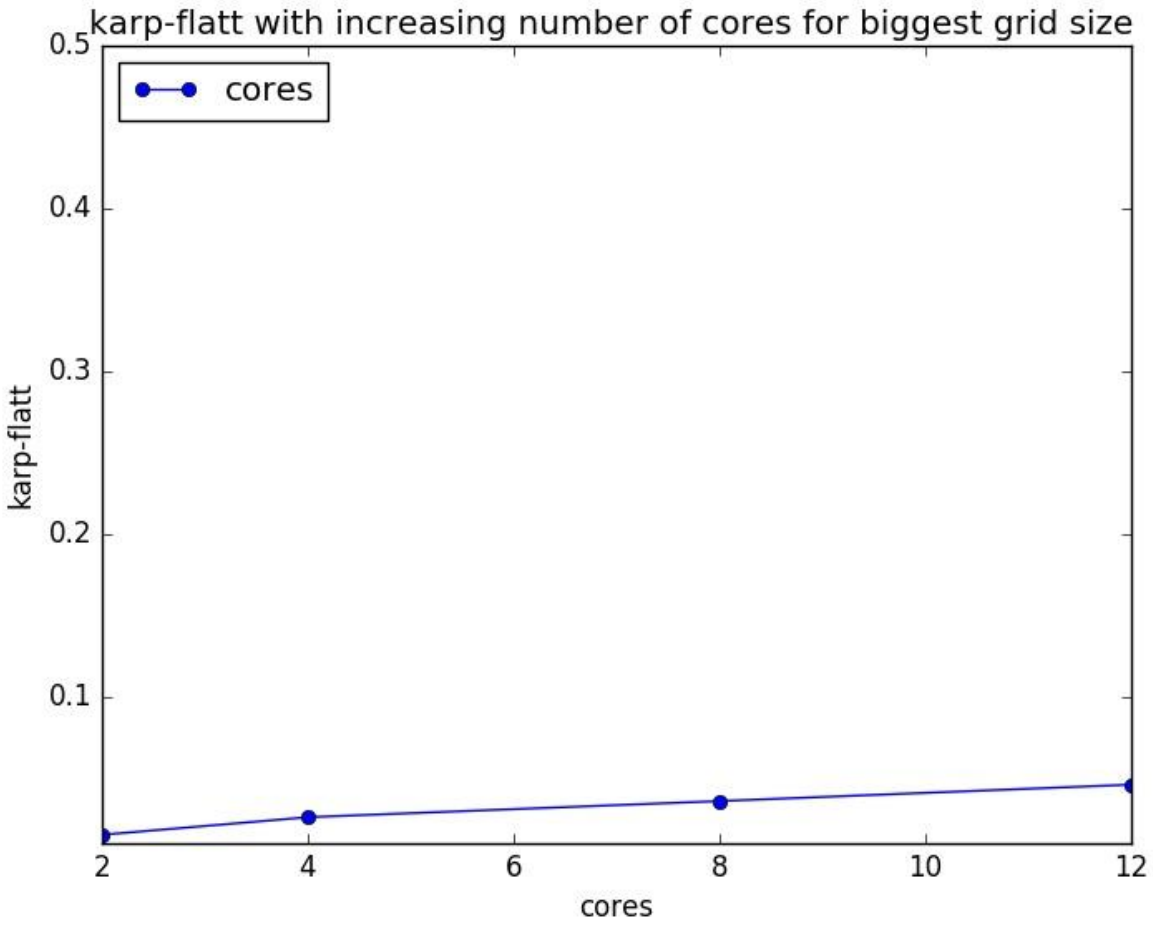
Benchmarking

Efficiency for Increasing Grid size



Benchmarking

Karp-Flatt for Increasing Grid size



Further work

1. Block update status
2. Bitwise calculating neighbors
3. Change list
4. HashLife (Ultimate improvement of life)

References

1. https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life Wikipedia- Game of life also GIF source.
2. https://en.wikipedia.org/wiki/Cellular_automaton Automaton
3. http://conwaylife.com/ Reading