CITS3402 – PROGRAMMING ASSIGNMENT 1

By Dylan Bell (2171951) And Ryan Hodgson (21969062)

1. OVERVIEW

Compile using gcc -fopenmp gol.c

The goal of this project was to demonstrate the efficiency that comes with parallel-programming compared to a traditional single threaded program. The program in question was "The Game Of Life" which was coded successfully in "C" as shown by the three graphs below. Each graph represents a snapshot of the game at a certain point of time. We can clearly see the progression of the game at each cycle.

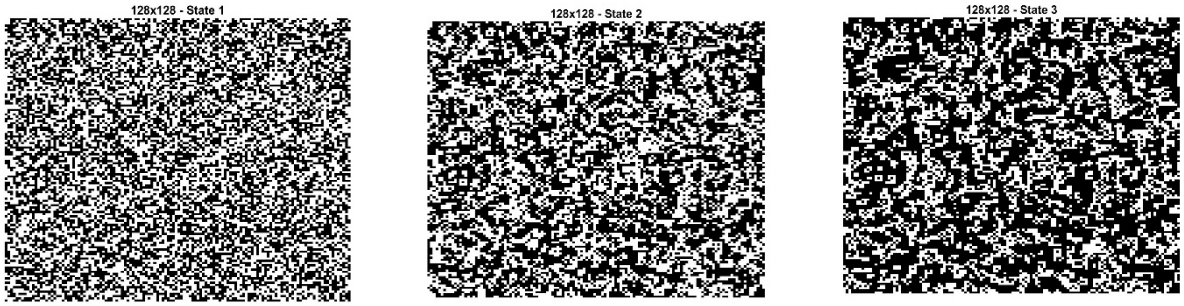

Figure 1.1 – Evolution Of "The Game Of Line"

Both methods of programming were implemented and timed. As discussed below significant improvements came with parallelising the game. Different experiments were trialled in hopes of improving the efficiency of the game even further.

2. EXPERIMENTS

A standard game was played on a board with dimensions 1024x1024, 100 generations/cycles and a single thread. With the introduction of parallel programming OpenMP automatically used the standard number of threads available at its disposal (in this case it was 4 threads).

Altogether there was three different programming implementations using various techniques.

- A non-parallel, standard implementation
- A parallelised version which used a simple "*#pragma omp parallel for*"
- A second parallelised version which took advantage of the OpenMP "*collapse()*" function

Three experiments were trialled, all measuring the time taken for each of the implementations.

- As the board dimensions are increased from 128x128 up to 2048x2048 how does the run time for each implementation change.

- For a fixed board dimension of 1024x1024 how does the increase in the number of generations/cycles change the run time for each implementation

- For the two parallelised implementations how does an increase in the number of threads change the run time for each implementation.

It was expected for the parallelising techniques to far outperform the standard, single threaded implementation.

3. RESULTS

Experiment 1

Our expectations were confirmed as shown by figure 3.1 and 3.2. In both cases the parallelised implementations took nearly half the time to complete the same program as a non-parallelised implementation.

GRID DIMENSIONS	128X128	256X256	512X512	1024X1024	2048X2048
NON-PARALLEL	0.11625	0.26034	1.04377	4.24257	16.8374
PAR. FOR	0.0376	0.13879	0.49994	2.17688	8.28562
SPEED UP	32.34%	53.31%	47.90%	51.31%	49.21%
PAR. COLLAPSE	0.03628	0.14092	0.50258	2.03989	8.09782
SPEED UP	31.21%	54.13%	48.15%	48.08%	48.09%

Figure 3.1 – Table Of Times For Experiment 1 In Seconds

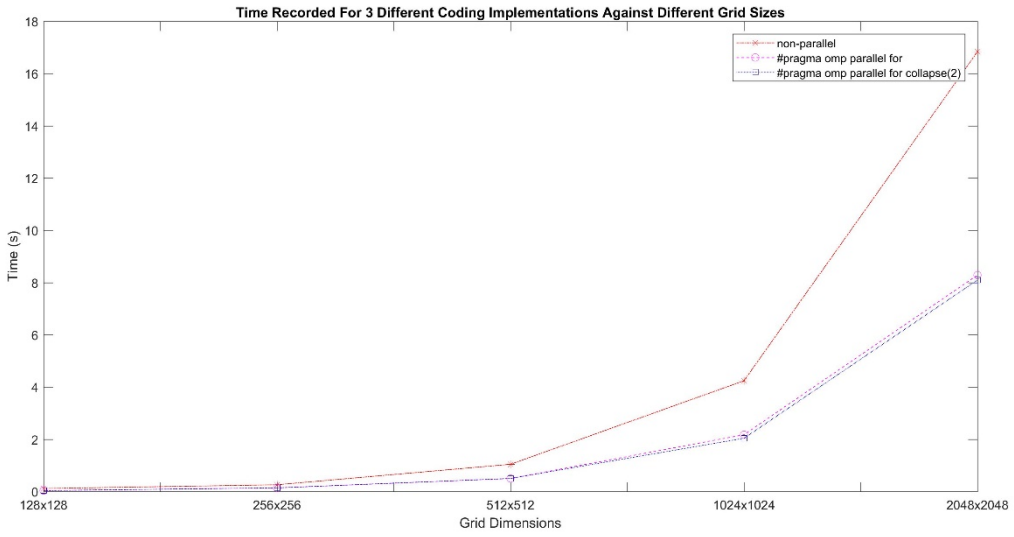

Figure 3.2 – Experiment 1 Graph

As displayed above the non-parallel implementation took a significantly longer time to process each grid. “Parallel For” and “Parallel Collapse” had similar times with no significant difference throughout the different grids. Each of these times were captured under the same system using the same number of threads (4) and generations (100). There’s a significant speedup because of a shared workload across four simultaneously threads as opposed to a single, sequentially executing thread.

Experiment 2

NUMBER OF CYCLES	100	500	1000	2000	4000
NON-PARALLEL	4.25251	21.00486	42.835	88.25753	169.9167
PAR. FOR	2.02748	10.87096	21.68274	48.69978	99.299
SPEED UP	47.68%	51.75%	50.62%	55.18%	58.44%
PAR. COLLAPSE	2.14105	10.45399	22.81439	48.50788	91.75643
SPEED UP	50.35%	49.77%	53.26%	54.96%	54.00%

Figure 3.3 – Table Of Times For Experiment 2 In Seconds

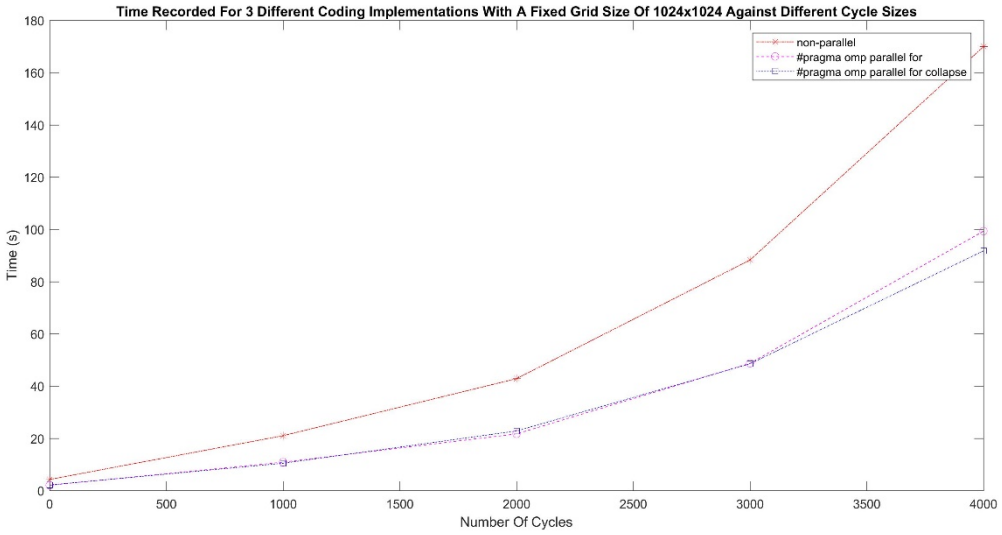

Figure 3.4 –Experiment 2

Similar to experiment 1, the non-parallelised implementation took a significantly longer period of time to complete. Again, the parallelised components had similar times and performed vastly better than the single threaded program. It is interesting to note how the “collapse” implementation starts to perform better after increasing the number of generations past 3000.

Experiment 3

NO OF THREADS	1	2	3	4	5	6	7	8	9	10
PAR. FOR	4.48815	2.49897	2.36094	1.99728	2.22554	2.28197	2.23403	2.00833	2.12212	2.2036
PAR. COLLAPSE	4.28452	2.5526	2.44229	2.12054	2.2129	2.14358	2.10576	2.10627	2.09616	2.07307

Figure 3.5 – Table Of Times For Experiment 3 In Seconds

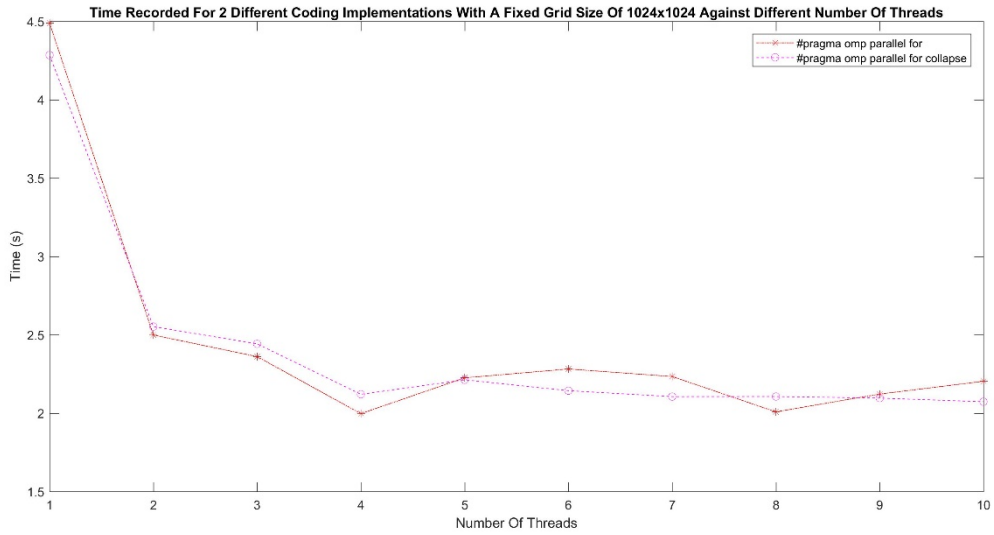

Figure 3.6– Experiment 3 Graph

For experiment 3 we wanted to test the effect of the number of cores on the time to execute. The computer we tested it on had 2 physical cores and 4 logical cores, as expected we saw a major speed up as we increased the number of threads up until 4 threads were being used. Once 5 threads were being used, there was a slowdown and as the number of threads increased further than that, no significant time change was noticed. One potential explanation to this outcome could be; when the number of threads is equal to the number of cores, it is straightforward. One core executes one thread, however as number of threads increase, there is context switching involved which adds a delay and increases the overhead. There are also other explanations, such as false sharing and cores not being fully utilised.

4. CONCLUSION

In all the experiments, the parallel implementation offered a significant speed up of around 50% over the sequential implementation. The 2 different parallel implementations performed with minimal difference, in some cases, one would outperform the other and vice-versa. This was explored thoroughly throughout different implementations and we saw that parallelisation offered a significant performance increase.