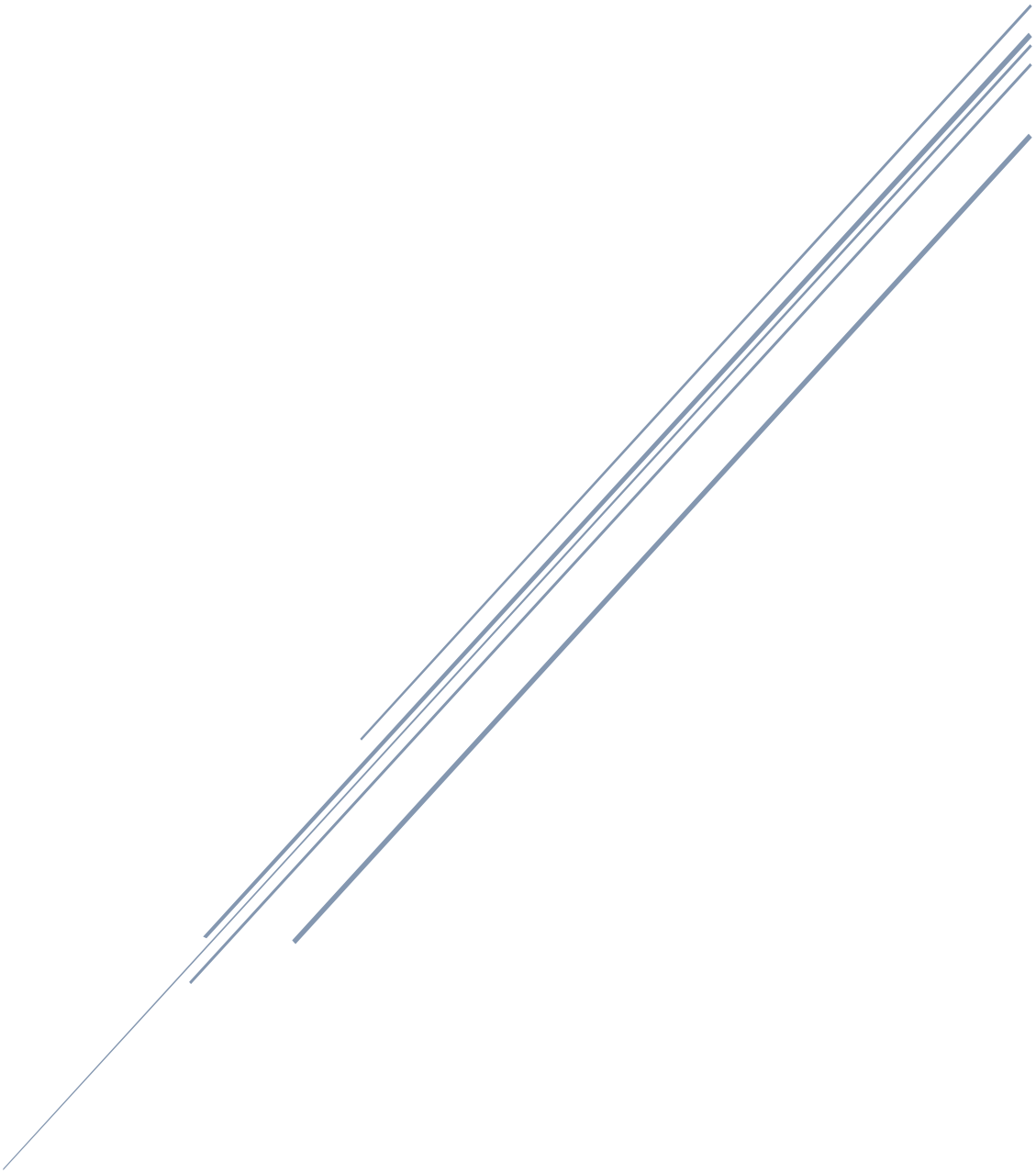
A MULTI THREADED GAME OF LIFE

Kieron Ho 20500057, Varun Jain 21963986



Assignment 1

CITS3402 High Performance Computing

Contents

How to compile our program

To compile the sequential code:

gcc -std=c99 -Wall -Werror -pedantic -o gameoflife gameoflife.c [array size]

example input: gcc -std=c99 -Wall -Werror -pedantic -o gameoflife gameoflife.c 128

To compile the openMP enabled code:

gcc -std=c99 -fopenmp gameopenMP gameopenMP.c [array size] [number of threads]

example input: gcc -std=c99 -fopenmp gameopenMP gameopenMP.c 128 2

Introduction

The game of life is a simulation played on a 2D board with a ruleset that determines if each grid cell is considered “alive” or “dead”. Each iteration, this board is updated with respect to the ruleset to determine if according to the current configuration the cell will be alive or dead. This update requires that each individual cell’s neighbour configuration be checked, as well as checking the current state of the cell, determining the state of the cell for the next iteration. This phenomenon introduces an interesting opportunity when it comes to the use of multi-threaded processing. Using OpenMP it is possible to use this multi-threading feature to improve the computation time between each game of life board update. Our code uses a randomly seeded configuration of living and dead cells as an initial state to make the most of the large board sizes.

OpenMP Implementation

The main section of our code that we used the OpenMP functionality is the loop for determining the fate of each cell ready for the next iteration. Due to the nature of OpenMP we were only able to divide the tasks by grid rows, meaning that each thread would be distributed whole rows each. The ability to separate out tasks by cells would provide a greater performance increase, however we were unable to implement this feature. It is predicted that providing more threads than there are rows would be a waste of resources, however the large size of game of life boards relative to the amount of reasonable threads means that this scenario is unlikely.

```
#pragma omp parallel for private(row, column, countNeighbours)
for(row = 0; row < arraySize; row++)
{
    for(column = 0; column < arraySize; column++)
    {
      countNeighbours = detect(array, row,column, arraySize);
      if((countNeighbours == 2 || countNeighbours == 3) && array[row][column] ==1)
      {
         newArray[row][column] = array[row][column];
      }
      if(countNeighbours == 3 && array[row][column] == 0)
      {
          newArray[row][column] = 1;
      }
        if(countNeighbours < 2 && array[row][column] == 1)
      {
          newArray[row][column] = 0;
      }
      if(countNeighbours > 3 && array[row][column] == 1)
```

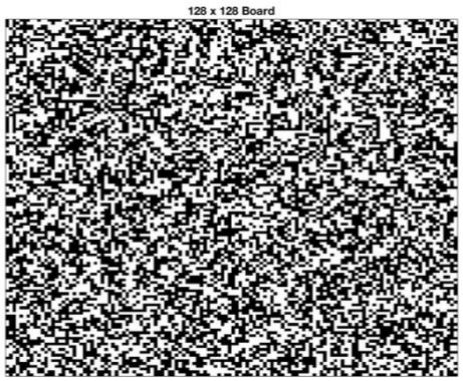
```
        {
            newArray[row][column] = 0;
        }
    }

}
```

Graphical Depiction of the Game of Life

Grid Size 128 X 128

Initial

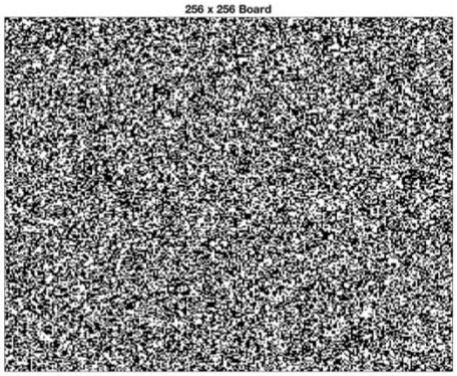


Step 2

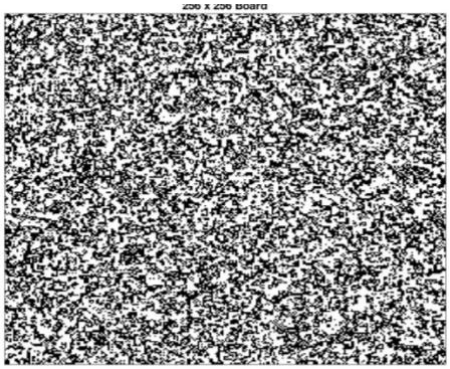
Step 3

Grid Size 256 x 256

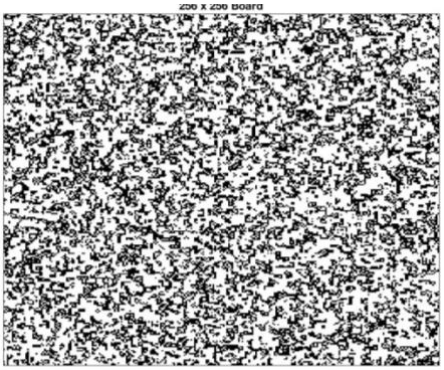
Initial



Step 2

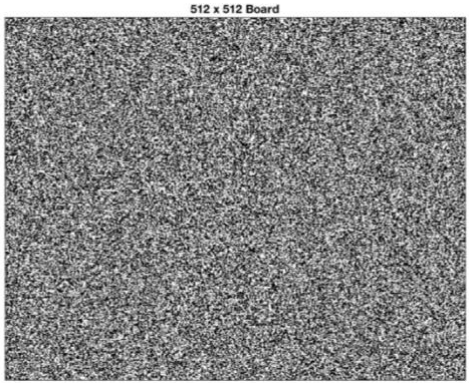


Step 3

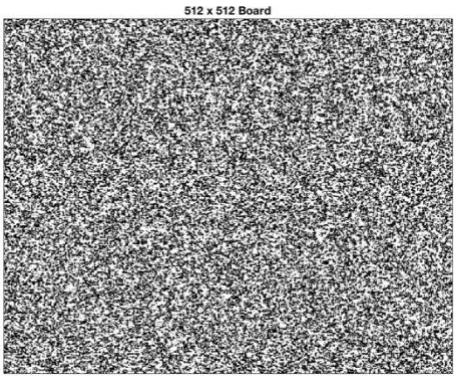


Grid Size 512 x 512

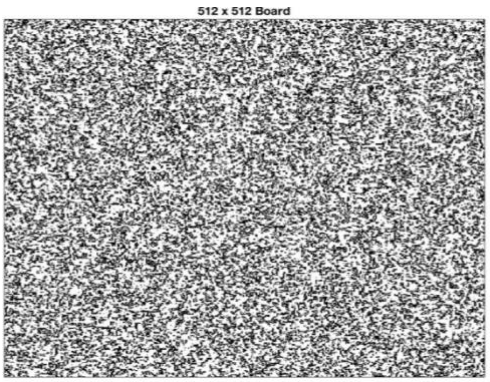
Initial



Step 2

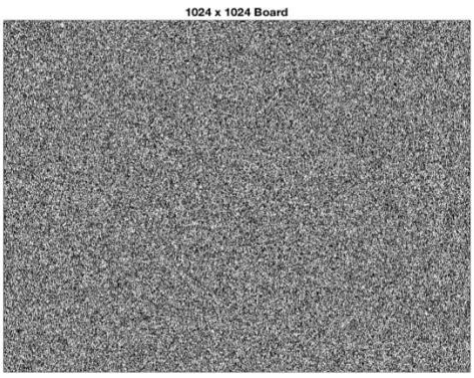


Step 3

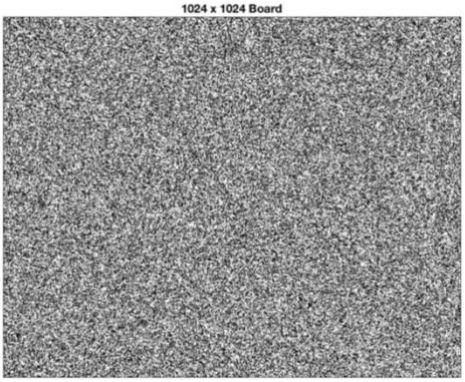


Grid Size 1024 x 1024

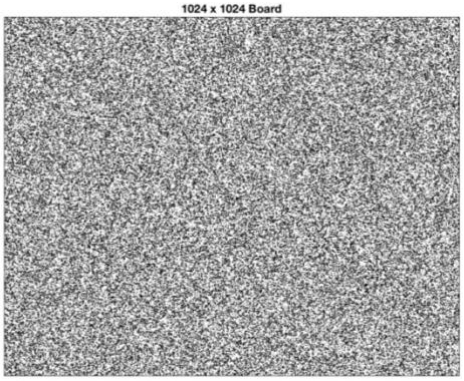
Initial



Step 2

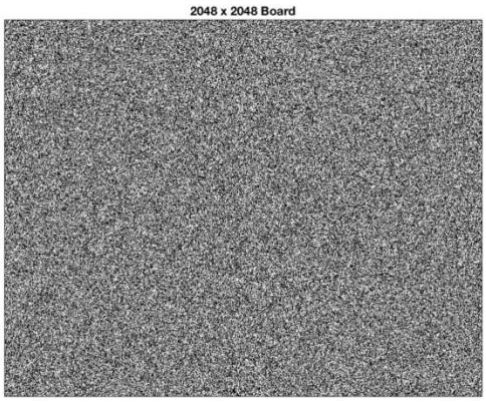


Step 3

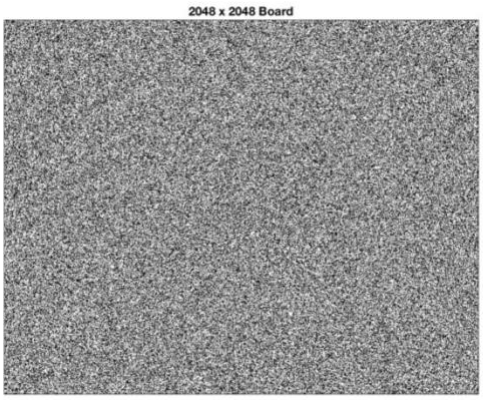


Grid Size 2048 x 2048

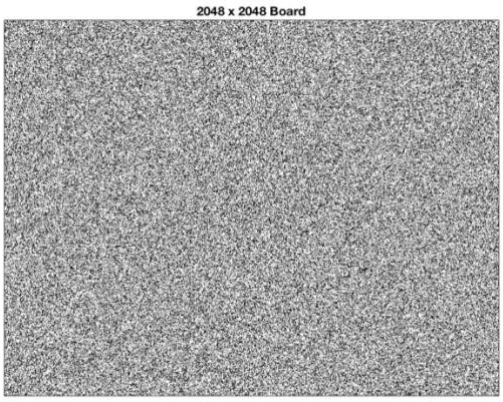
Initial



Step 2



Step 3



Timing Comparisons

Experiment Specifications

These experiments were run on Windows 7 computers with the following system specifications:

- Processor: Intel® Core™ i7-4790S CPU 3.2-GHz
- RAM: 8.00 GB
- System type: 64-bit Operating System

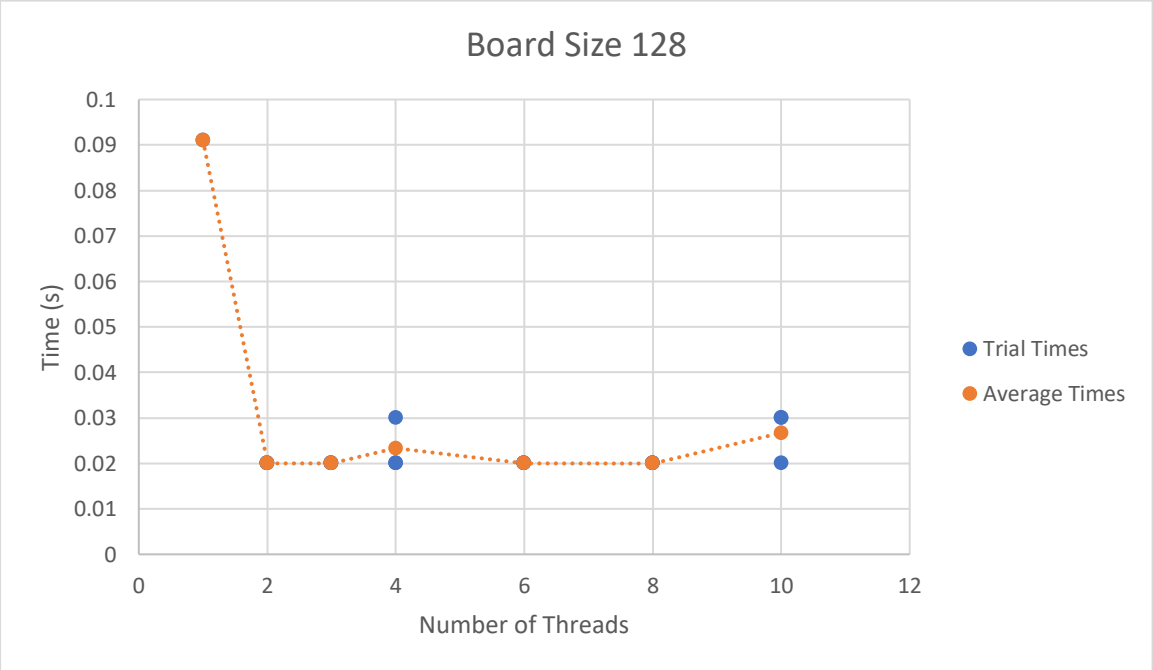
The experiments were run with differing size “Game of Life” boards, as well as differing processing conditions. These were predominantly whether the code was run with OpenMP enabled or disabled. While disabled, the process can be considered to be running sequentially, while OpenMP tests will be run using a thread counts of 2, 3, 4, 6, 8 and 10. These values are the result of the game running through 100 steps.

Table of average timed results

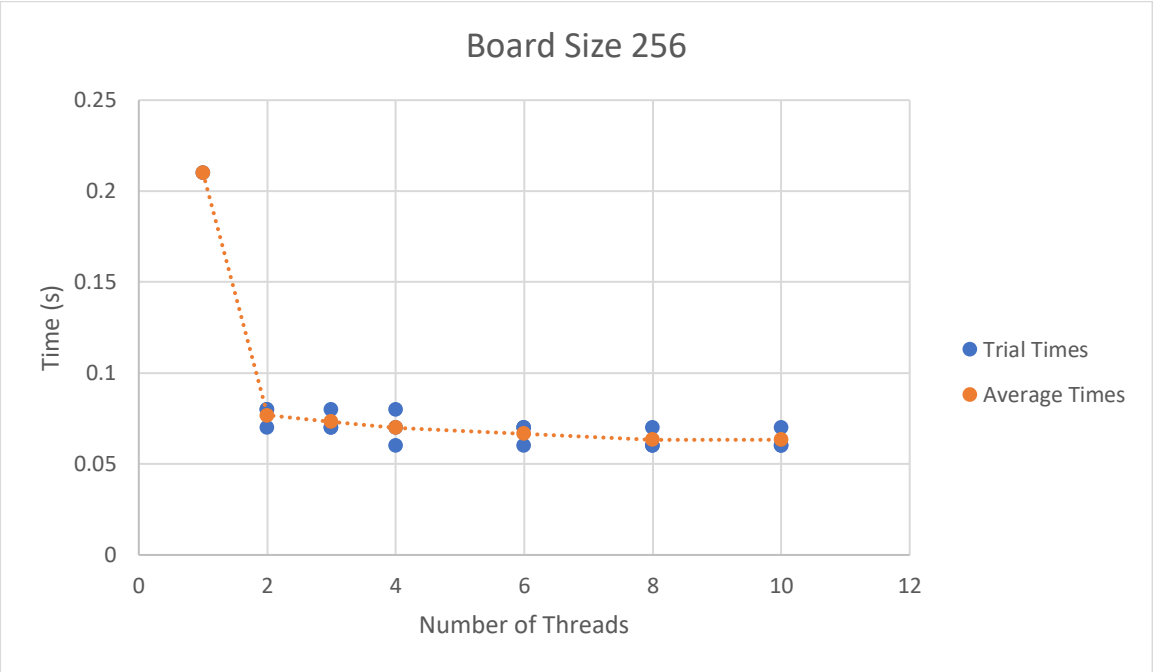
	Board Size				
Thread Count	128 X 128	256 X 256	512 X 512	1024 X 1024	2048 X 2048
Sequential	0.091 sec	0.20993 sec	0.549984 sec	2.039938 sec	8.239772 sec
2	0.019999 sec	0.076665 sec	0.283658 sec	1.180964 sec	4.533864 sec
3	0.019999 sec	0.073331 sec	0.22666 sec	0.973637 sec	3.596892 sec
4	0.023333 sec	0.069998 sec	0.219993 sec	0.743311 sec	3.276902 sec
6	0.019999 sec	0.066665 sec	0.233326 sec	0.796976 sec	3.040576 sec
8	0.02 sec	0.063331 sec	0.249992 sec	0.843308 sec	2.976575 sec
10	0.026663 sec	0.063331 sec	0.219993 sec	0.846641 sec	3.096573 sec
Improvement	78%	68%	60%	64%	63%

Individual Size Results

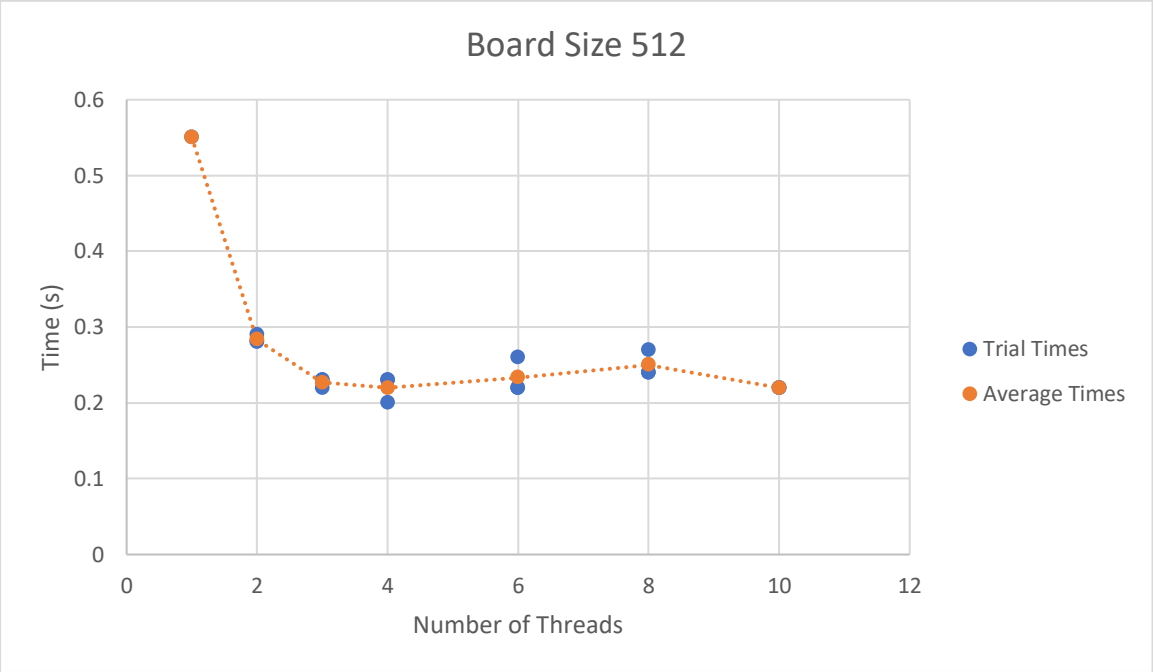
128 X 128



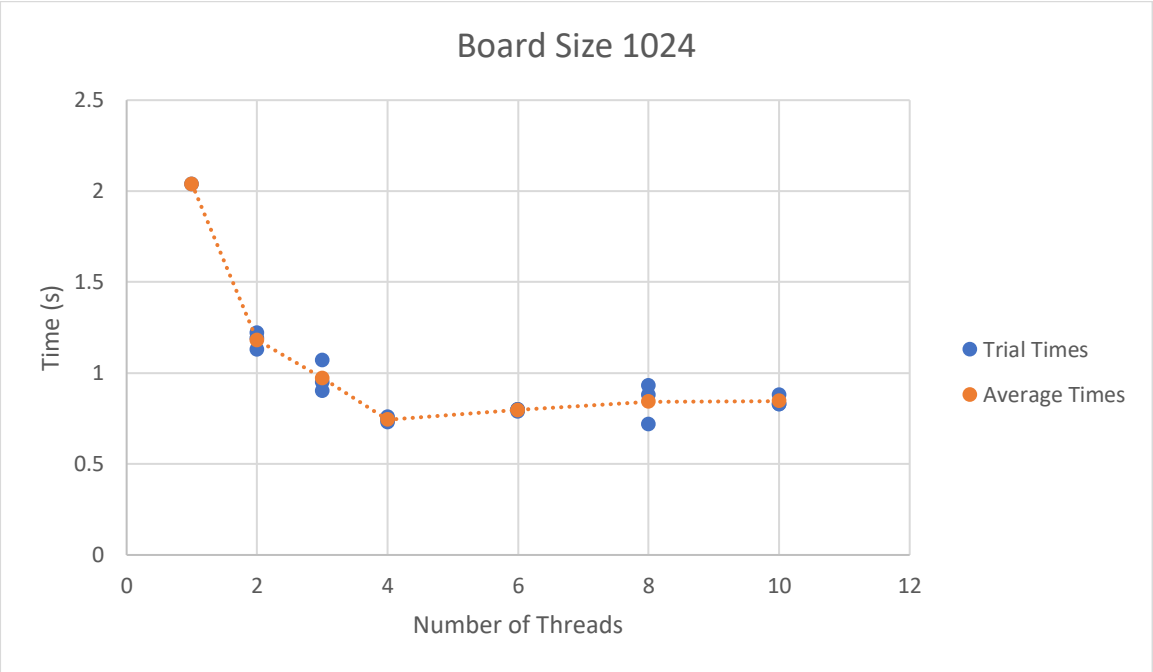
256 X 256



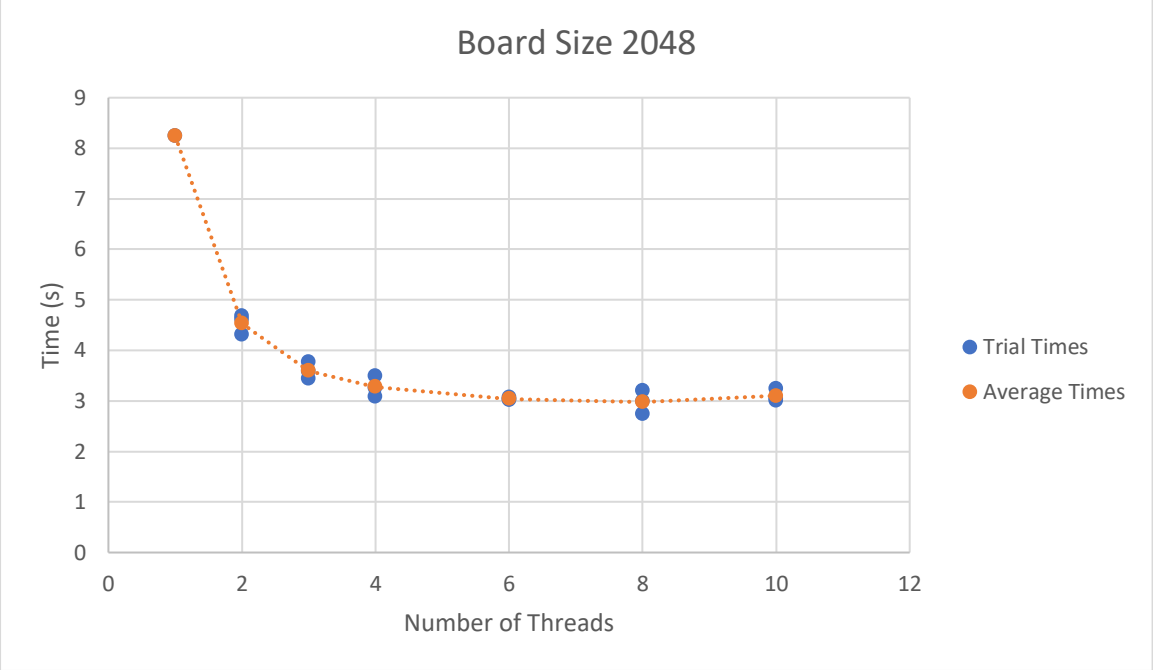
512 X 512



1024 X 1024



2048 X 2048



Conclusion

As is evident through our experimentation and testing, OpenMP is an excellent method of increasing computational times when multithreaded calculations may be implemented into the process. However, based on our experiments there are a few interesting points to take away from this:

1. At all board sizes the introduction of multiple threads immediately increases the speed of the program
2. At shorter time spans there are clear increments of time steps that prevent a high degree of precision. At higher time spans this is not such an issue
3. All board sizes feature a particular thread count where the speedup of the program no longer grows with the thread count. This is most likely due to the fact that the threads will share the same processing resource, and their concurrency will eventually cause a bottleneck in speed increase.

At best, multithreading benefits the processing times in the following ways:

- Board Size 128 increased speed by 78% using 2 threads
- Board Size 256 increased speed by 68% using 6 threads
- Board Size 512 increased speed by 60% using 4 threads
- Board Size 1024 increased speed by 64% using 4 threads
- Board Size 2048 increased speed by 63% using 8 threads