MSTISR001
CSC2002S
Assignment 1
Parallelism

Introduction

In this assignment, the task was to parallelize a serial version of the Monte Carlo algorithm. The output of the parallel program will then be compared to the serial as well as the execution time. From this a conclusion will be made upon whether or not parallelization is worth the extra work.

Method

Java's Fork/join framework is used to implement the parallelization. The compute() method reduces the size of the bigger array to smaller arrays that can run on different cores at the same time. The chunks of the array will run on threads, each thread performs a separate search operation on the same terrain, and the global minimum is tracked along with the coordinates. Two separate parallel programs were created both implementing a similar approach as mentioned but executed differently. The first version(D4) divides the work into 4 equal parts for threads to work on, the second one is a more traditional(TP) way of using a sequential cut-off to separate work into threads.

Divide into 4(D4):

This version was created first, the reasoning behind the attempt was to test whether a constant speedup of approximately 4 could be obtained no matter the size of the work. This didn't work for one machine(PC) but did somewhat work in the other machine(Mac). The reasoning behind this could be due to how the different architectures divide threads among cores. There were some issues of array indexes being out of bounds(either -1 or +1 more than the index). The methods visited(int x, int y) and find_valleys() were changed in Search and TerrainArea, respectively, to accommodate this.

Traditional Parallelism(TP):

This version was created to attempt to achieve a better speedup time, this only worked for the first machine(PC) and not for the second one(Mac). The D4 program outperformed the TP program on the Mac in every run consistently. No issues were found in terms of bounds or anything else so no changes were made to the existing classes.

One change that was made to both of them was that Direction was made into a separate class for issues with the make file.

Validation:

To validate the correctness of the program, the results of the serial and parallel programs were compared to each other using the Rosenbrock function. Although all programs including the serial one would sometimes show instability, both programs produced consistent minimum values and locations across multiple runs.

Serial Output(Tested on PC):

```
Run parameters

Rows: 5000, Columns: 5000

 x: [0,000000, 1500000,000000], y: [0,000000, 1500000,000000]

 Search density: 1,000000 (25000000 searches)

Time: 2997 ms

Grid points visited: 15803144  (63%)

Grid points evaluated: 94806117  (379%)

Global minimum: 10000 at x=0,0 y=0,0
```

D4 Output(Tested on PC):

```
Run parameters

Rows: 5000, Columns: 5000

x: [0,000000, 1500000,000000], y: [0,000000, 1500000,000000]

Search density: 1,000000 (25000000 searches)

Time: 1467 ms

Grid points visited: 11901735  (48%)

Grid points evaluated: 47980498  (192%)

Global minimum: 10000 at x=0,0 y=0,0
```

TP Output(Tested on PC):

```
Run parameters

 Rows: 5000, Columns: 5000

 x: [0,000000, 1500000,000000], y: [0,000000, 1500000,000000]

 Search density: 1,000000 (25000000 searches)

Time: 1403 ms

Grid points visited: 5484484  (22%)

Grid points evaluated: 17053836  (68%)

Global minimum: 10000 at x=0,0 y=0,0
```

Machine Architectures

Machine A(PC):	**Machine B(Mac):**
Operating System – Windows 11	**Operating System – macOS Ventura**
CPU – 11th Gen Intel Core i7	**CPU – Apple M1**
GPU – NVIDIA GeForce RTX 3050 Ti	**GPU – Apple M1**
RAM – 16GB	**RAM – 8GB**
Storage – 512GB SSD	**Storage – 256GB**
Cores – 8	**Cores – 8**
Logical Processors – 16	**Logical Processors – 16**

Benchmarking

Two scenarios are measured in the assignment, Speedup vs no. of Searches and Speedup vs Grid Size. First, the serial program is run on each machine 10 times, this is done so that the lowest time be chosen. After that the same is done for both parallel versions, and shortest time is recorded. Depending on which scenario is being tested, the search density or the gird sizes are changed and the whole process is repeated. The speedup is then calculated and plotted against the work.

Difficulties

Other than the issues with index bounds in D4, the only other issue was a race condition that caused a miscalculation with the number of points visited and evaluated. This was fixed by making the variables static. This was done so that the shared data can be accessible to all threads since some threads weren't able to update it, however it can affect thread safety. A better what to implement this would be either with synchronization or Atomic variables, the reason why this was not considered is because it is not important in this assignment.

Results

PC:

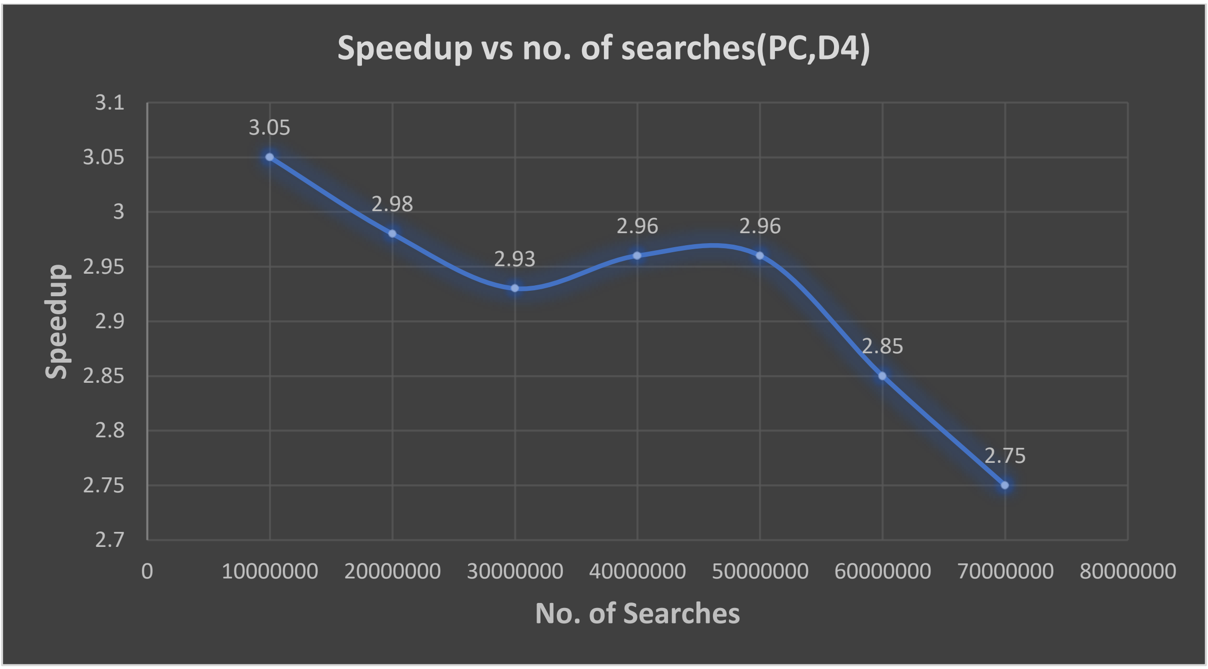

Fig 1.1

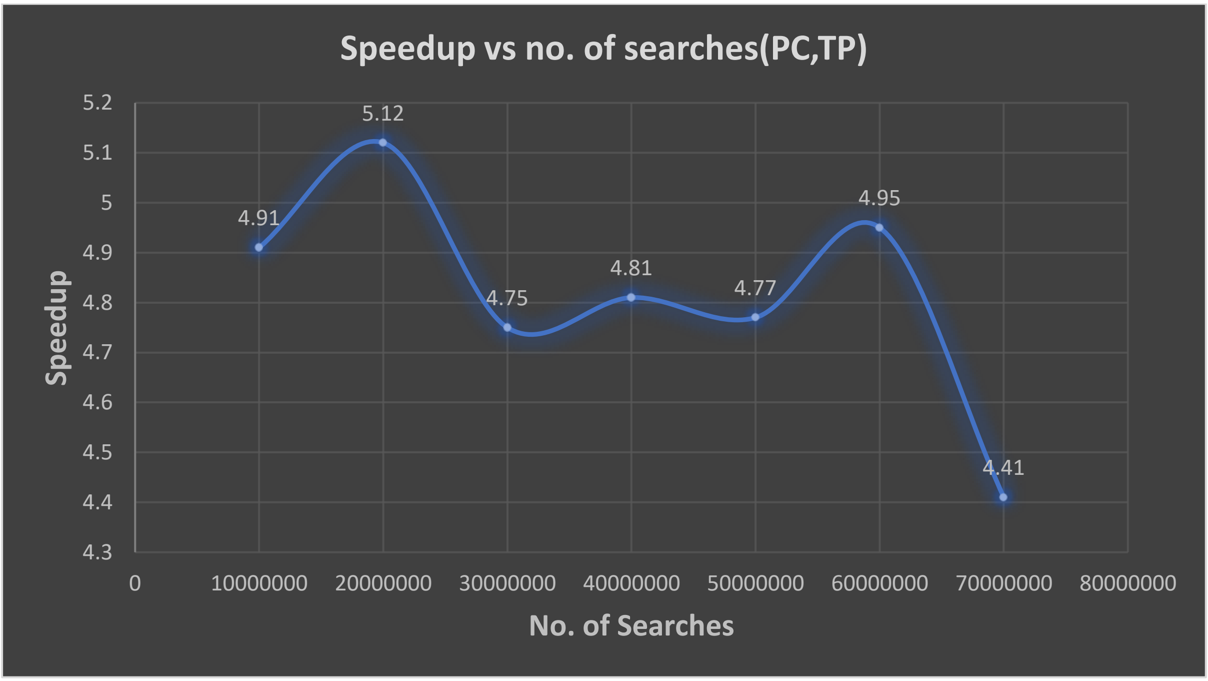

Fig 1.2

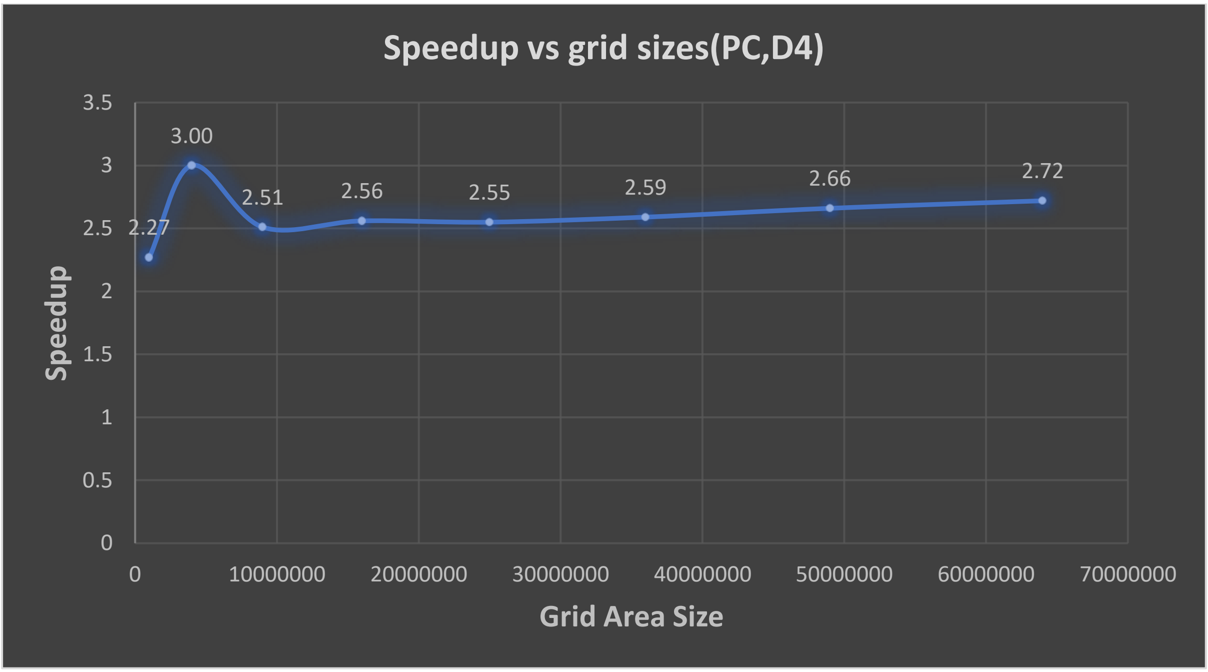
Speedup vs grid sizes(PC,D4)
Speedup
Grid Area Size
3.5
3
2.5
2
1.5
1
0.5
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000
2.27
3.00
2.51
2.56
2.55
2.59
2.66
2.72

Fig 2.1

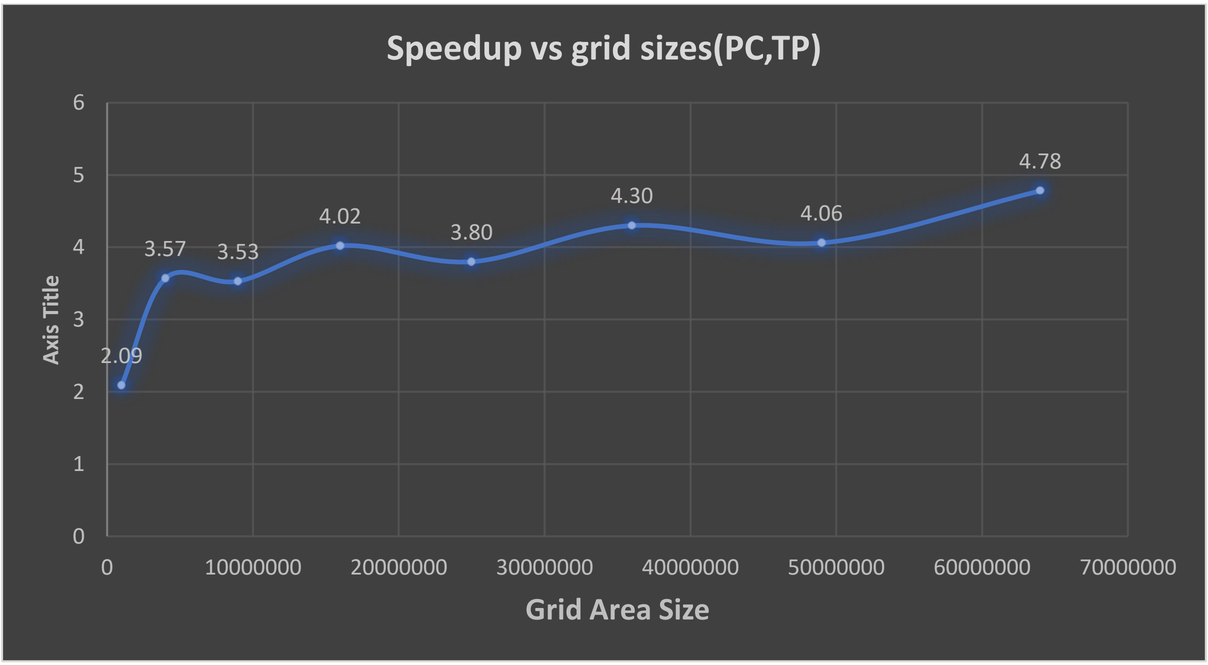
Speedup vs grid sizes(PC,TP)
Axis Title
Grid Area Size
6
5
4
3
2
1
0
0
10000000
20000000
30000000
40000000
50000000
60000000
70000000
2.09
3.57
3.53
4.02
3.80
4.30
4.06
4.78

Fig 2.2

Mac:

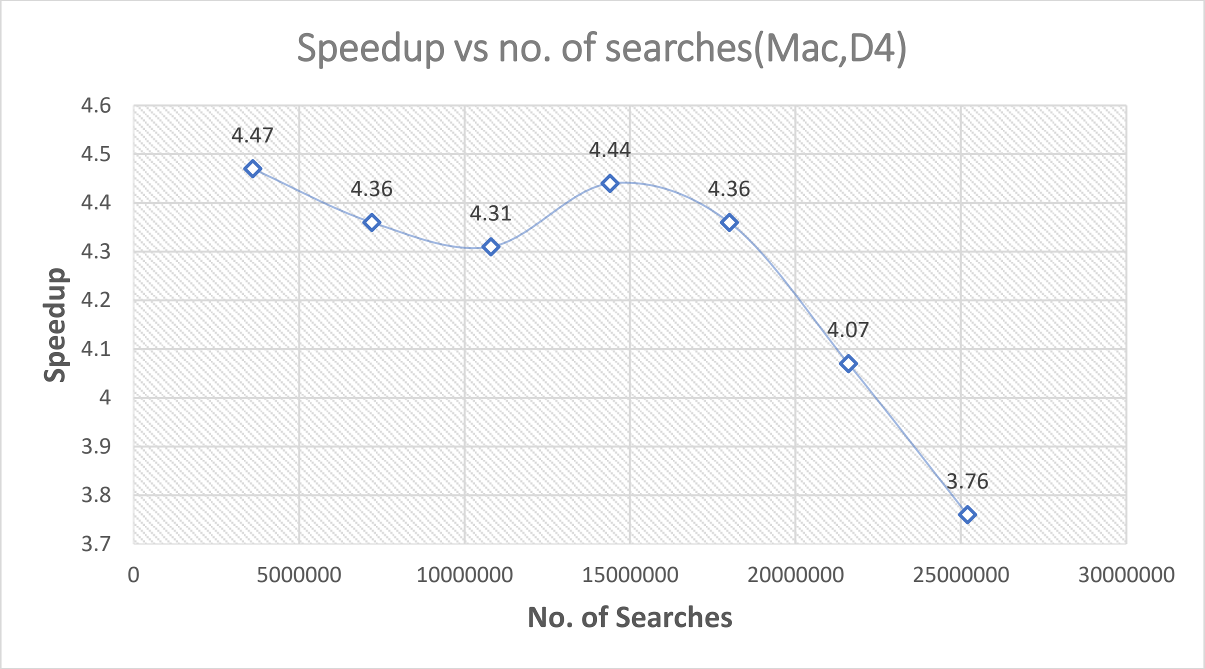

Fig 3.1

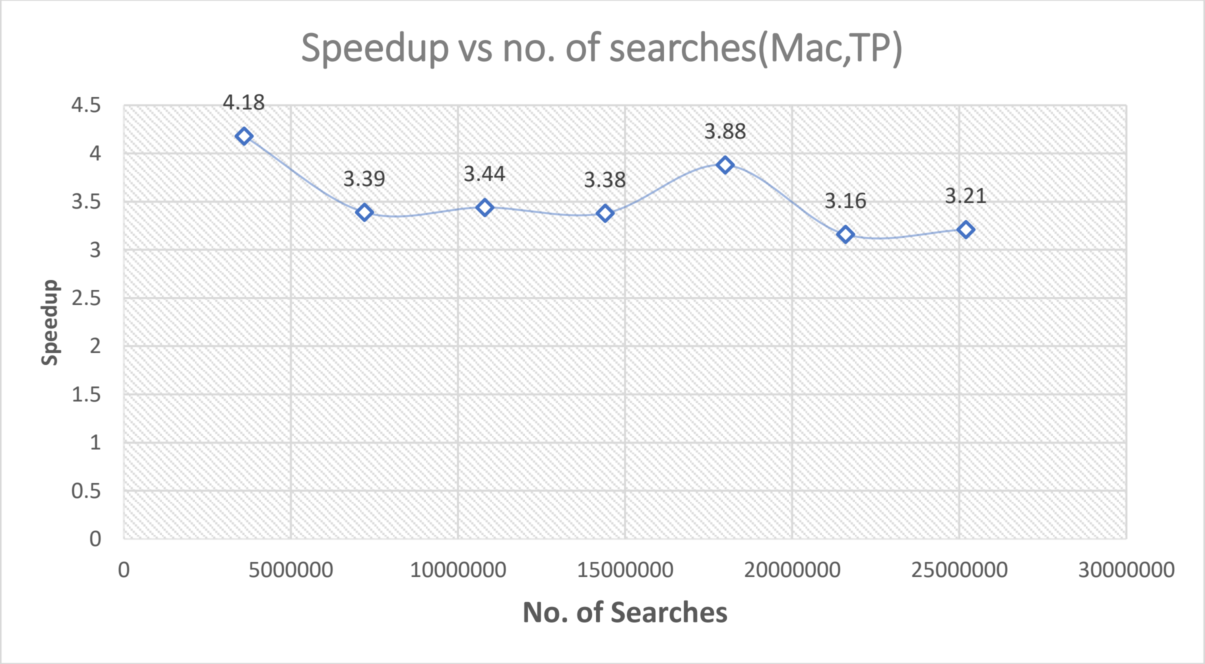

Fig 3.2

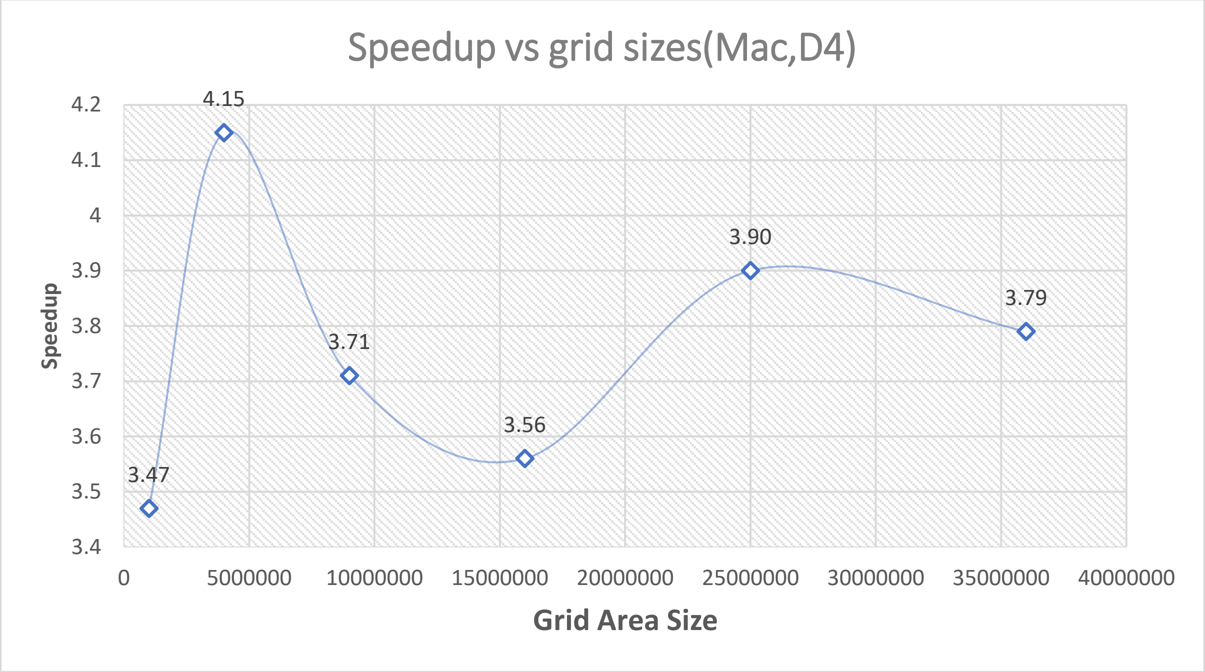
Speedup vs grid sizes(Mac,D4)
Speedup
Grid Area Size
4.2
4.1
4
3.9
3.8
3.7
3.6
3.5
3.4
0
5000000
10000000
15000000
20000000
25000000
30000000
35000000
40000000
3.47
4.15
3.71
3.56
3.90
3.79

Fig 4.1

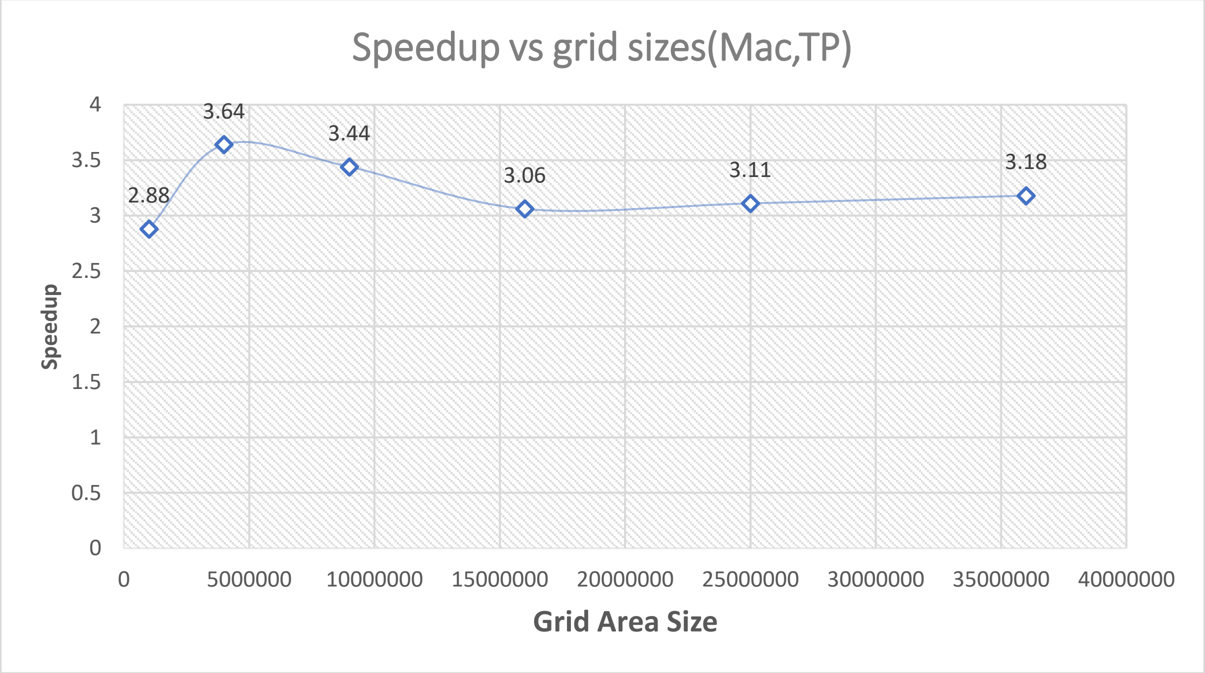
Speedup vs grid sizes(Mac,TP)
Speedup
Grid Area Size
4
3.5
3
2.5
2
1.5
1
0.5
0
0
5000000
10000000
15000000
20000000
25000000
30000000
35000000
40000000
2.88
3.64
3.44
3.06
3.11
3.18

Fig 4.2

Discussion & Conclusions

It is interesting to see that the two programs perform oppositely to each other depending on the architecture. It was found that D4 was more unstable than TP when running on the PC, this could due the lack of proper thread safety. Factors such as lower work loads had a negative effect on the stability of all programs, including the serial one, although not as common as the parallel counterparts. I'm unsure as to why that is, it could be due to the random element in the Monte Carlo algorithm but that does not explain why it is more stable with bigger workloads. Another interesting thing was that no matter what the cutoff was for TP(whether it was 10,1000 or 5000) it ran with similar execution times throughout.

The range of that the parallel programs work well are from grid sizes 1000 to 10000(maximum search density at 100000 is 0.8 for PC) for both programs, although the Mac couldn't handle higher workloads, the highest it could go is around 6000 x 6000 x 1(36000000 searches), higher than that and a heap error would occur.

The highest speedup obtained in the testing was 5.12 on the PC while running TP. This value is not much higher than the other values of speedup when running other workloads on the PC using TP, the average speedup was around 4.82(**Fig 1.2**), there were some anomalies where runtime was slower than most or when the output is unstable. The parallel programs performed as expected and ran faster than the serial program, the speedup reaches between 3.0 and 4.0+ constantly.

In conclusion, it is worth it to use parallel to speed up the runtime, it will produce good results but for more accurate results, some synchronization is needed to achieve more stable results.