Group 4: Ian Spiegel, Chris Smith
10/15/19
Project 2
CSC 345

In our project we implemented all the possible requirements within the project description. Each requirement implementation is described below.

The project takes in input from a file named input.txt that should contain a 9x9 grid of numbers that simulate a sudoku board. Input.txt contains 81 numbers total and 9 numbers in each row and column. We use an fopen function in order to open a file labeled exactly “input.txt” and then read that file line for line to find the number inside it. The code will look specifically for a 9x9 grid of numbers, and any spaces or new-lines between the numbers in input.txt will be excluded from any work in the code. The code will also print out exactly what is included in input.txt to show the board state back to the user.

Our program takes the sudoku board from input.txt and searches it using threads to figure out and validate whether the sudoku board contains a solution or not. We print out the board state and the file the board state is coming from. After this we check the numbers with the threads with either a single thread through the columns and rows, or multiple threads for the columns and rows depending on what the user requests to use. Typing in 1 after main will provide the user to run the program using a single thread to navigate the rows and columns and depending on the values of the board, the program will return either YES, the solution exists, or NO, a solution does not exist. After each YES or NO, the time will be announced to the user of how long the execution took to figure out the result.

In our program we have two different ways of validating the sudoku board, either with one thread for all columns and one thread for all rows (1) OR 9 threads for 9 columns and 9 threads for 9 rows in the sudoku board (2). To execute this type either ./main 1 for the method with one thread for the row and columns, and ./main 2 for the second process with 9 threads for 9 columns and 9 threads for 9 rows in the sudoku board. We recognize argv1 as either a 1 or 2 and any other value after ./main will not allow the program to run. So if we have just ./main or perhaps ./main 3, the program will not run. We also take this argv1 to see which methods and functions we will use to validate the board with conditional if statements. Both methods check the subgrids, rows, and columns, but the difference is how the program runs depending on choosing the method. 1 will use only one thread to check the rows and one thread for the columns, but 2 will check each row with one thread individually, so we have 9 threads for the columns and 9 threads for the rows. 9 threads are always used for the subgrid.

Our Statistical analysis and conclusion of the experiment comparing and contrasting method 1 and 2 is below:

Statistical Experiment:

For our statistical experiment, we compared the execution times for running our code on the VM provided to us that uses linux, and on our own personal MacBook Pro laptop. Our experiment used the same input.txt which was a NO solution, and we ran 50 trials for both computers and recorded data for various different types of statistics(mean, mode, standard deviation, etc.).

Data for Method 1 and 2 Plotted on TCNJ Macbook

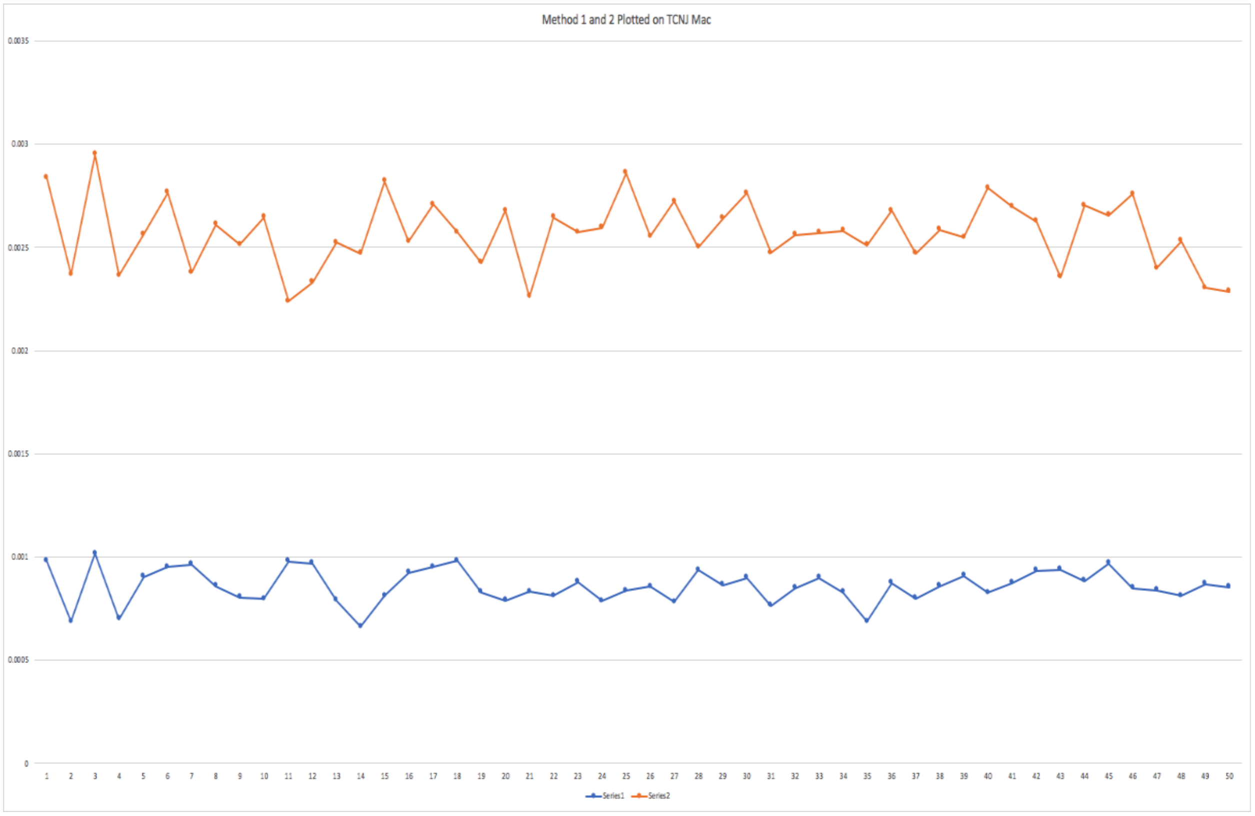
Statistic	Method 1(s)	Method 2(s)
Standard Deviation	.0000816	.0001700
Mean	.0008590	.0017100
Median	.0008575	.0017145
Mode	.0006880	.0017100
Minimum	.0006610	.0012600
Maximum	.0010160	.0020230

All data points from TCNJ Mac (50 execution times)

Method 1	**Method 2**
0.000979	0.001857
0.000688	0.001681
0.001016	0.001934
0.000699	0.001664
0.000903	0.001661
0.000951	0.001814
0.000963	0.001417
0.000859	0.001753
0.000804	0.00171
0.000797	0.00185
0.00098	0.00126
0.000968	0.001364

0.00079	0.001734
0.000661	0.00181
0.000811	0.00201
0.000924	0.001605
0.000952	0.001755
0.000982	0.001591
0.00083	0.001597
0.00079	0.001889
0.000832	0.001429
0.000813	0.001832
0.000878	0.001695
0.000788	0.001807
0.000837	0.002023
0.000857	0.001696
0.000782	0.001942
0.000936	0.001564
0.000865	0.001776
0.000899	0.001863
0.000764	0.00171
0.000849	0.001711
0.000898	0.001672
0.000829	0.001751
0.000688	0.001823
0.000873	0.001806
0.000799	0.001671

0.000858	0.001728
0.000909	0.00164
0.000826	0.001963
0.000874	0.001824
0.000934	0.001692
0.000939	0.001418
0.000883	0.00182
0.000968	0.001689
0.00085	0.001909
0.000839	0.001559
0.000812	0.001718
0.000869	0.001434
0.000855	0.001432



Data for Method 1 and 2 Plotted on Personal Computer with VM Linux

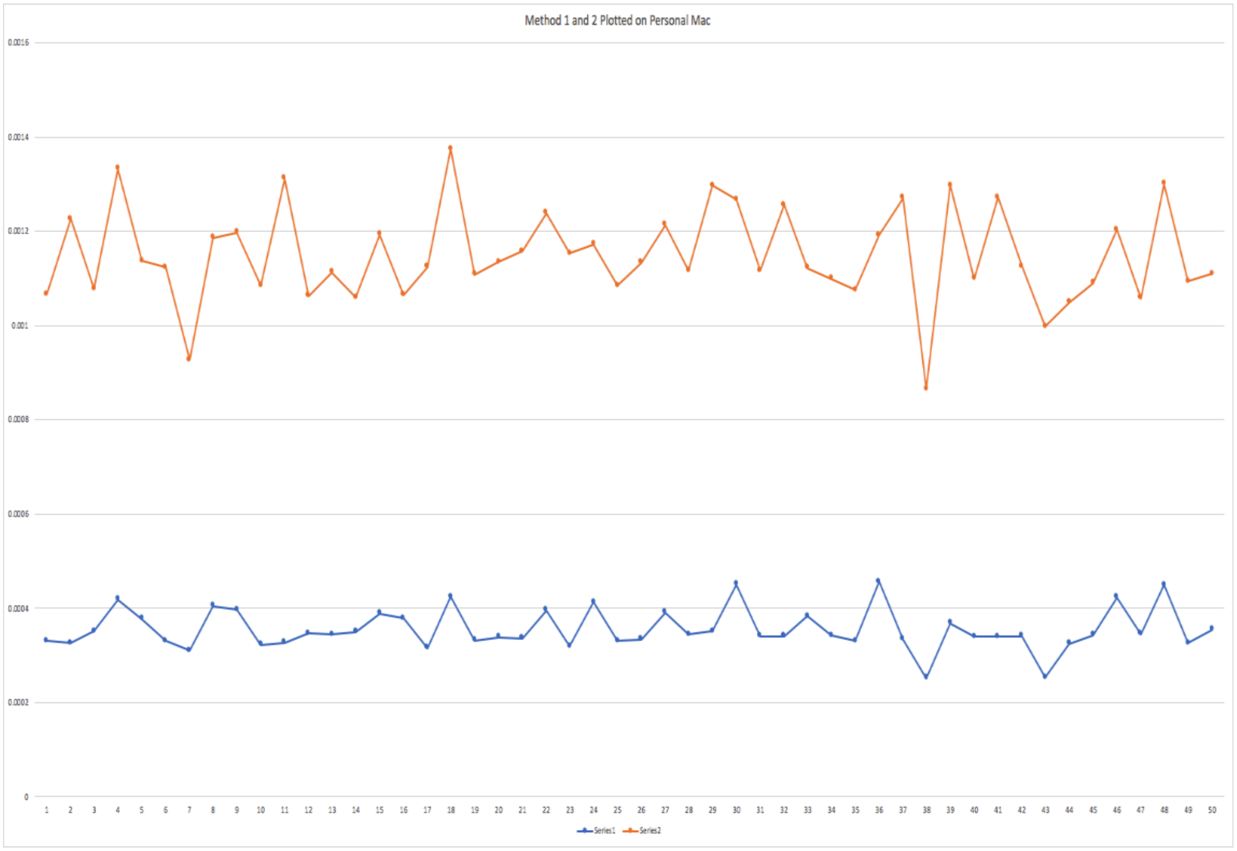
Statistic	Method 1(s)	Method 2(s)
Standard Deviation	.0000433	.0000815
Mean	.0003553	.0007933
Median	.0003425	.0007765
Mode	.0003310	.0008000
Minimum	.0002520	.0006130
Maximum	.0004560	.0009850

All data points from Personal Computer using VM (50 execution times)

Method 1	**Method 2**
0.000331	0.000735
0.000326	0.0009
0.000351	0.000727
0.000418	0.000914
0.000377	0.00076
0.00033	0.000793
0.00031	0.000617
0.000405	0.000781
0.000397	0.0008
0.000323	0.000762
0.000327	0.000985
0.000346	0.000717
0.000344	0.000769
0.00035	0.000709
0.000389	0.000804
0.000378	0.000687
0.000316	0.000808
0.000424	0.000951
0.000332	0.000777
0.000339	0.000796
0.000336	0.000821
0.000396	0.000843
0.000319	0.000834

0.000413	0.000759
0.000331	0.000754
0.000334	0.0008
0.000391	0.000823
0.000344	0.000773
0.000351	0.000946
0.000451	0.000816
0.000341	0.000776
0.000341	0.000915
0.000383	0.000739
0.000342	0.000757
0.000331	0.000744
0.000456	0.000736
0.000334	0.000937
0.000252	0.000613
0.000368	0.000929
0.00034	0.000759
0.00034	0.000932
0.000341	0.000785
0.000253	0.000745
0.000325	0.000724
0.000343	0.000747
0.000423	0.00078
0.000345	0.000713
0.000449	0.000851

0.000326	0.000768
0.000355	0.000755



Conclusion:
We ran 50 independent trials on the Mac in the TCNJ lab room and on our personal laptop's VM. Here is the data we found with line graphs of our data points for all 50 trials. Method 1 ran slower on both computers. We would assume that the first method would be slower and less efficient than the second method since the second method implements nine separate threads where each of them checks one column. And Method 1 only creates one thread that checks all nine columns. However, due to our implementation in our code, the throughput of Method 2 would be slower. On the personal computer with VM our standard deviation was smaller, which means our data on our personal laptop was more precise and consistent than the Mac at TCNJ. All other statistical methods gave smaller numbers for our personal laptop, this might be because the Macs at TCNJ do not run as efficiently as our personal laptops. So, based on our results, we found that through both the VM and Mac execution, the code ran on average slower with method 2 than 1. We suspect this might be due to the implementation of method 2 within our code. We used a for loop to create 9 different threads in parallel to each other in the code for columns, but we did this multiple times as well because we also test the subgrid and rows of the board. The first method although only using one thread for the rows and one for the

columns, uses a specific function that makes it run more efficiently called memset(). This function allows us to reinstate the same thread through the same void function multiple times to run through all the columns or rows within the board. This allows us to avoid using a for loop to create multiple threads for the rows or columns. Because we avoid these for loops, our performance becomes much faster, and because of this, this is why we theorize that method 1 goes faster than method 2. Both move pretty swiftly in both the Mac and VM, but on average, method 1 takes roughly a little less than half the run-time of method 2. It’s possible with a different implementation method 2 would be faster, but with ours method 2 is slower. VM uses multiple cores, but roughly the same result occurs when executing on Mac, so it appears that the number of courses used during execution does not impact the overall results of the code’s method speed comparison. The Mac performance was slower than the VM performance, but this might be due to clutter on the Mac machine. Mac’s use 4 physical cores, but 8 logical cores. Because of this we believe cores don’t impact the comparison of speed and it’s rather Mac’s lack of drivers in OS X. Mac is typically a bit slower than linux VM due to a lack of drivers for the custom Apple hardware. All in all, the data points towards Method 1 running faster than Method 2.