COP290 Assignment 1: Task 2 Report

Shubham Sarda (2018TT10958) Ankit Kumar (2020CS10323)

In this subtask, we aim to improve the efficiency (runtime) of the matrix multiplication function we implemented in subtask 1. We've tried to do the same using

1) Libraries such as mkl and openblas 2) Multithreading using pthreads

We've created 5 source (.cpp) files and corresponding header (.h) files to ensure independence and organization within functions employing different algorithms:

1) main.cpp
2) a1task2_pthread.cpp
3) a1task2_mkl.cpp
4) a1task2_openblas.cpp
5) a1task2_naive.cpp

We've used incremental build using Makefile in order to build an executable by the name of "yourcode.out". Incremental build ensures only modified source files (and those for which it is a dependency) are compiled in a recursive manner.

$make: Used for compilation to create "yourcode.out"

$make clean: Used to remove all ".o" and ".out" files

Source file "main.cpp" is the one where we handle different commands from the command line/terminal. Activation, pooling and probability function are unchanged from subtask1. These are implemented in a1task2_naive.cpp and are executed using commands same as those used in subtask1.

Activation: `$./yourcode.out activation type inputmatrix.txt outputmatrix.txt`

Pooling: `$./yourcode.out pooling type inputmatrix.txt stride outputmatrix.txt`

Probability: `$./yourcode.out probability type inputvector.txt outputvector.txt`

For Fully Connected (matrix multiplication and addition), one additional command has been added to select the type of implementation to be used

Sample command line for fully connected layer output:

```
$./yourcode.out fullyconnected inputmatrix.txt weightmatrix.txt biasmatrix.txt outputmatrix.txt implementation
```

Output = Input * Weight + Bias, where, '*' = matrix multiplication, '+' = element wise addition.

Here "implementation" can be one of {"mkl", "openblas", "pthread", "naive"}.

a) MKL: In case implementation is "mkl", functions employed in "a1task2_mkl.cpp" are used.

Function signature: `vector<vector<float>> fc_output_mkl(string input, string weights, string bias)`

The above function is used to return fully connected output when input matrix, weight matrix and bias matrix are stored in the files "input", "weights" and "bias". These file names are passed as arguments to the function. In this function, a mkl library function "`cblas_sgemm(…)`" is used which takes the matrices in flattened format (1D array) and returns the output. The flattened output is again converted to 2D vector and returned.

b) Openblas: In case implementation is "openblas", functions employed in "a1task2_openblas.cpp" are used.

Function signature: vector<vector<float>> fc_output_openblas(string input, string weights, string bias)

The above function is used to return fully connected output when input matrix, weight matrix and bias matrix are stored in the files "input", "weights" and "bias". These file names are passed as arguments to the function. In this function, an openblas library function "cblas_sgemm(…)" is used which takes the matrices in flattened format (1D array) and returns the output. The flattened output is again converted to 2D vector and returned.

c) Pthread: In case implementation is "pthread", functions employed in "a1task2_pthread.cpp" are used. Function signature: vector<vector<float>> fc_output_pthread(string input, string weights, string bias).

This function takes input matrix, weight matrix and bias matrix from the files given in the arguments and then performs matrix multiplication and addition using multithreading.

Here pthread library has been used thread creation and synchronization. Helper function "fc_layer_fn_using_pthread(…)" is used to create, join and exit the pthreads. Another helper function "mul_ fn(…)" implements the subroutine performed by each thread. More specifically, it creates each row of the output matrix. For ensuring proper synchronization of each thread, mutex function has been used for locking and unlocking of the threads. We've experimented with different number of threads, but found best results (least time and max cpu usage) when number of threads is equal to number of rows in input matrix, i.e. one thread to create every row of output.

d) Naive: In case implementation is "naive", the O(n^3) algorithm used in subtask1 is employed. It is implemented in the file "a1task2_naive.cpp".
Function signature: vector<vector<float>> fc_output_naive(string input, string weights, string bias)

In addition, we've added another functionality to create files for making the boxplots.

Plot File Creation: $/.yourcode.out plot matrix_size num_runs outputfile implementation

1. matrix_size: This is the matrix size on which we want to create plot files.

2. num_runs: This is the number of times we want to run matrix multiplication and addition on a fixed matrix size.

3. outputfile: This is the output file in which we write all the runtimes along with mean and standard deviation

4. implementation: It can one of {"mkl", "openblas", "pthread", "naive"} to select the type of algorithm to be employed.

The function runs the matrix multiplication and addition operations "num_runs" times by creating randomized input, weights and bias matrices of size ("matrix_size" X "matrix_size"). Runtime for each run is stored in the file with name "outputfile". Chrono library is used to measure the runtimes in microseconds.

Function Signatures for different implementations:

1) void create_plot_files_mkl(int matrix_size, int num_runs, string outputfile)
2) void create_plot_files_openblas(int matrix_size, int num_runs, string outputfile)
3) void create_plot_files_pthread(int matrix_size, int num_runs, string outputfile)
4) void create_plot_files_naive(int matrix_size, int num_runs, string outputfile)

In order to create multiples plot files at once:

Plot File Creation: $/.yourcode.out plot_all num_runs implementation

Here we run “num_runs” iterations for matrix sizes:
{10,20,30,40,50,60,70,80,80,100,150,200,250,300,350,400,450,500,550,600,650,700,750}

Matrix size increases in gaps of 10 from 10 to 100 and in gaps of 50 from 100 to 750.

The files are named as “plot_{implementation}_{matrix_size}_{num_runs}.txt”. For eg:

“plot_mkl_100_50.txt”

Correctness: To ensure correctness, we also create a “truth” output matrix in Mkl, Openblas and Pthread functions. The truth matrix is created from naïve algorithm. The outputs from these three are compared to truth matrix to check correctness.

Error handling: Error checking has also been implemented as done in subtask1.

a) A check is applied to verify if the number of arguments inputted are same as that required to execute the task (argc==7 in case of fully connected output). It also verified that files given in command line exists or not. Further it is ensured that “implementation” is one of {“mkl”, “openblas”, “pthread”, “naive”}.

b) A check is applied to verify if dimensions of input and weight matrices are of the form (A x B) and (B x C) respectively to ensure matrix multiplication is possible. Further dimensions of bias matrix are verified to ensure it is (A x C) for element wise addition to be possible.

In case of any error, program is terminated nicely.

Comparative Study using Boxplots

We've used gnuplots to create boxplots for different matrix sizes and implementations. We've run 50 iterations for each matrix size to create the plots.

We observed that in general standard deviations were quite high due to several outliers. Further on comparing the runtimes, we 've concluded the following order for efficiency:

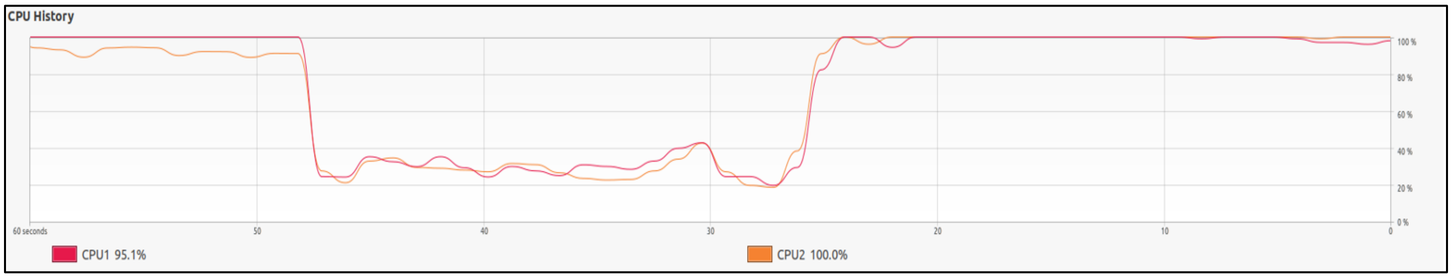
MKL>Openblas>Pthread>Naïve

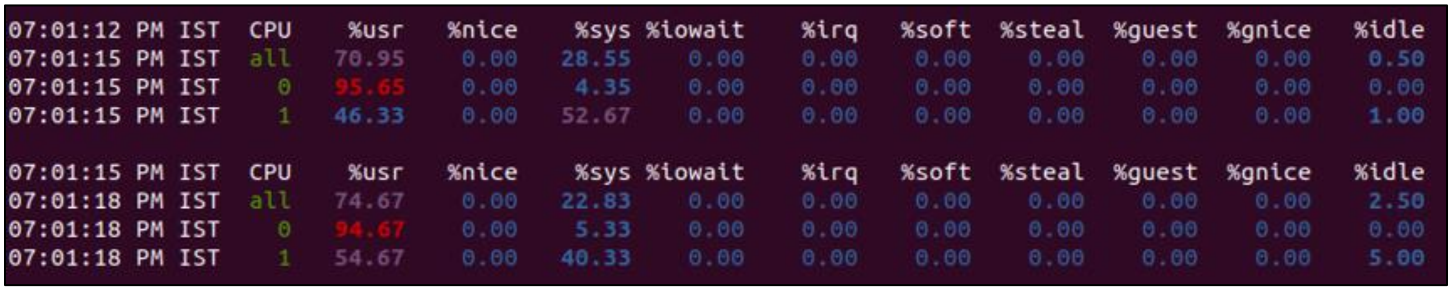
MKL and Openblas are much faster than pthread and Naïve. Further Mkl outperforms openblas slightly.

This is also reflected in the cpu usage for these implementations as shown below. MKL and Openblas utilize both cores to a maximum extent. Pthread also utilizes both cores but % utilization is low. Naïve only utilizes one core.

Note: For the line graphs, observe the time of past 20 seconds, since that is when the program execution started.

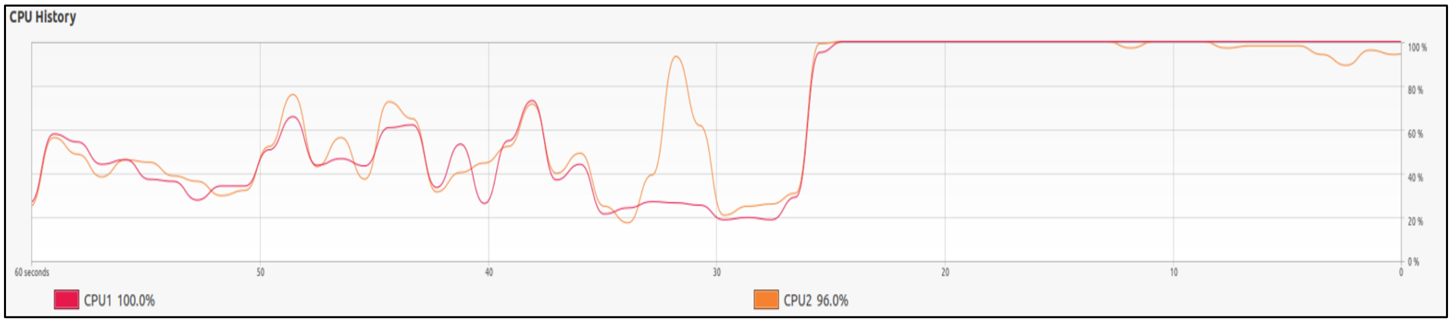
MKL:

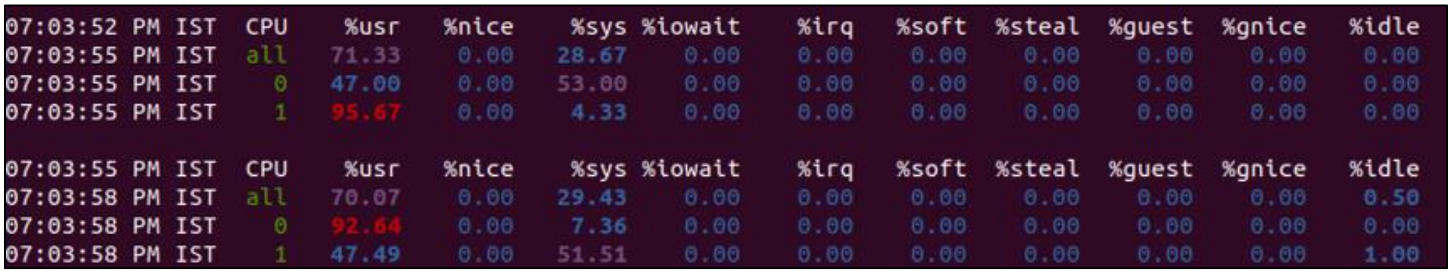




```
07:01:12 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
07:01:15 PM IST  all   70.95    0.00   28.55    0.00    0.00    0.00    0.00    0.00    0.00    0.50
07:01:15 PM IST    0   95.65    0.00    4.35    0.00    0.00    0.00    0.00    0.00    0.00    0.00
07:01:15 PM IST    1   46.33    0.00   52.67    0.00    0.00    0.00    0.00    0.00    0.00    1.00

07:01:15 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
07:01:18 PM IST  all   74.67    0.00   22.83    0.00    0.00    0.00    0.00    0.00    0.00    2.50
07:01:18 PM IST    0   94.67    0.00    5.33    0.00    0.00    0.00    0.00    0.00    0.00    0.00
07:01:18 PM IST    1   54.67    0.00   40.33    0.00    0.00    0.00    0.00    0.00    0.00    5.00
```

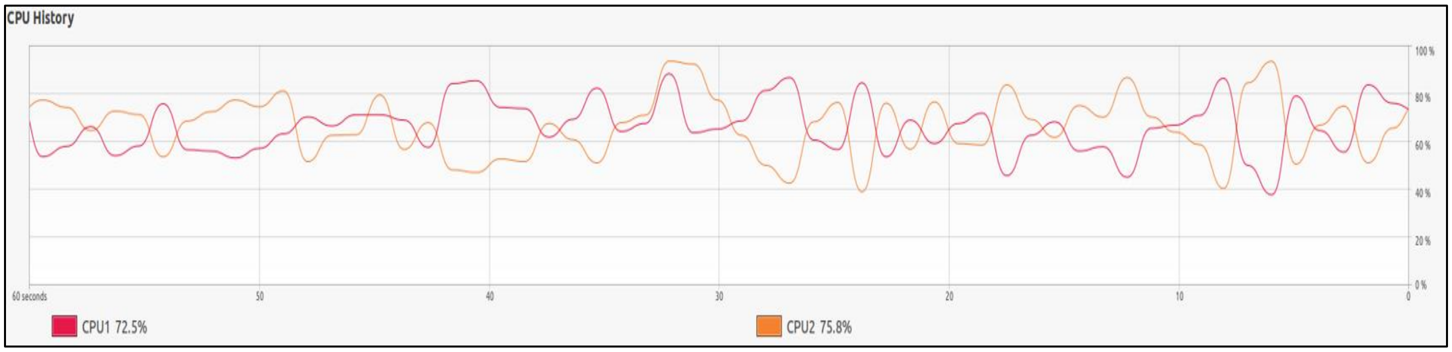
OpenBlas:

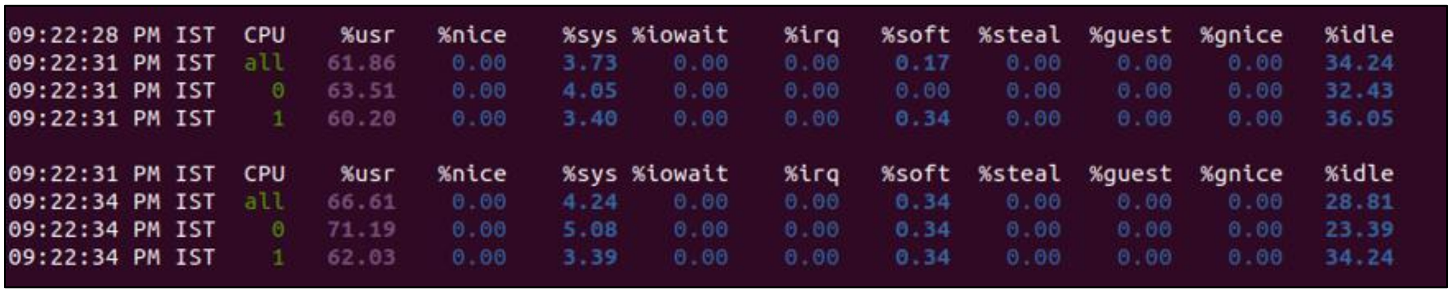




```
07:03:52 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
07:03:55 PM IST  all   71.33    0.00   28.67    0.00    0.00    0.00    0.00    0.00    0.00    0.00
07:03:55 PM IST    0   47.00    0.00   53.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
07:03:55 PM IST    1   95.67    0.00    4.33    0.00    0.00    0.00    0.00    0.00    0.00    0.00

07:03:55 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
07:03:58 PM IST  all   70.07    0.00   29.43    0.00    0.00    0.00    0.00    0.00    0.00    0.50
07:03:58 PM IST    0   92.64    0.00    7.36    0.00    0.00    0.00    0.00    0.00    0.00    0.00
07:03:58 PM IST    1   47.49    0.00   51.51    0.00    0.00    0.00    0.00    0.00    0.00    1.00
```

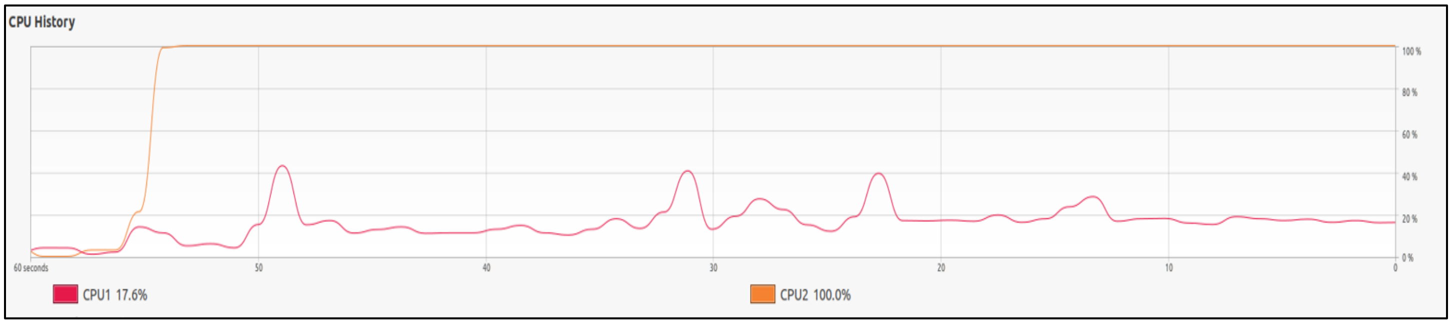
Pthread

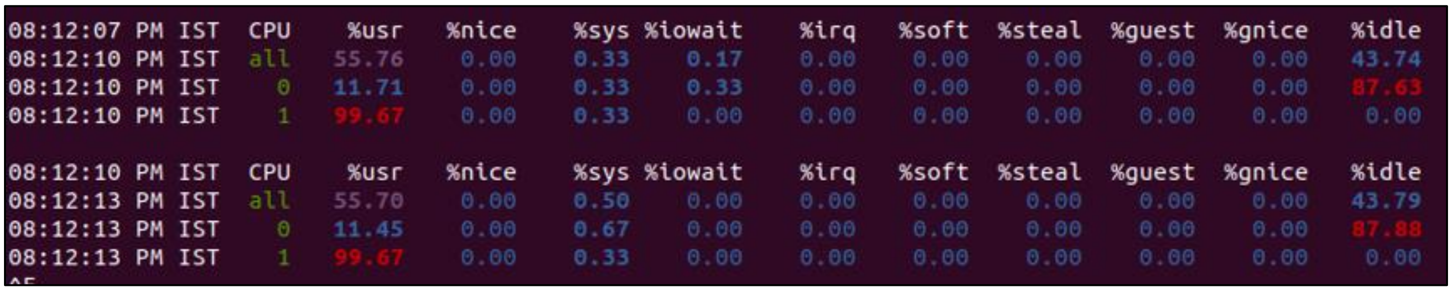




```
09:22:28 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
09:22:31 PM IST  all   61.86    0.00    3.73    0.00    0.00    0.17    0.00    0.00    0.00   34.24
09:22:31 PM IST    0   63.51    0.00    4.05    0.00    0.00    0.00    0.00    0.00    0.00   32.43
09:22:31 PM IST    1   60.20    0.00    3.40    0.00    0.00    0.34    0.00    0.00    0.00   36.05

09:22:31 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
09:22:34 PM IST  all   66.61    0.00    4.24    0.00    0.00    0.34    0.00    0.00    0.00   28.81
09:22:34 PM IST    0   71.19    0.00    5.08    0.00    0.00    0.34    0.00    0.00    0.00   23.39
09:22:34 PM IST    1   62.03    0.00    3.39    0.00    0.00    0.34    0.00    0.00    0.00   34.24
```

Naïve:





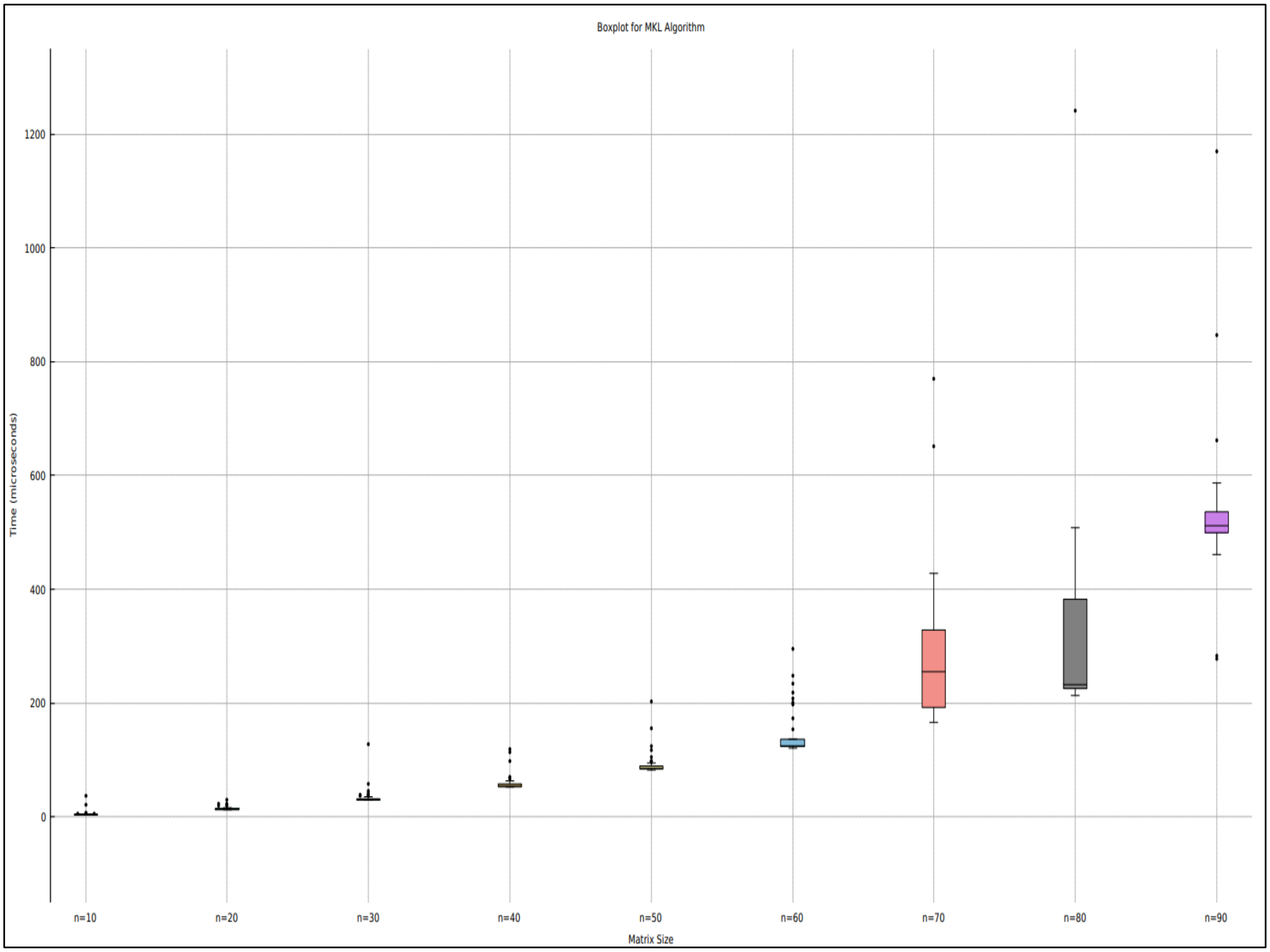
```
08:12:07 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:12:10 PM IST  all   55.76    0.00    0.33    0.17    0.00    0.00    0.00    0.00    0.00   43.74
08:12:10 PM IST    0   11.71    0.00    0.33    0.33    0.00    0.00    0.00    0.00    0.00   87.63
08:12:10 PM IST    1   99.67    0.00    0.33    0.00    0.00    0.00    0.00    0.00    0.00    0.00

08:12:10 PM IST  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
08:12:13 PM IST  all   55.70    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00   43.79
08:12:13 PM IST    0   11.45    0.00    0.67    0.00    0.00    0.00    0.00    0.00    0.00   87.88
08:12:13 PM IST    1   99.67    0.00    0.33    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

For a more detailed analysis, we've shown some plots, along with sde and mean for some matrix sizes. For plotting, we've used several scripts. The script just needs to be altered in line 15 and 25 to modify xtics and file name in order to plot different files. We've created 4 directories "mkl_plots", "openblas_plots", "pthread_plots" and "naive_plots". In these directories, there are multiple plot files (names start with "plot" and created using the "create_plot_files(…)" functions mentioned above) containing runtimes and several script files (names start with "script"). Script files are named according to range of matrix sizes they plot. For instance, in the "mkl_plots" folder, file "script_700_750" creates a plot for matrix range 700-750 for the mkl implementation. We've also added pdfs format of the plots shown below. The script on running, doesn't directly save the image/pdf since it requires some manual zooming and cursor shifting to get all plots in one screen, hence we've directly saved the pdfs of plots after this manual processing. Steps for plotting:

1) Open terminal in directory containing script and plot files
2) $gnuplot: opens gnu setup in terminal
3) $load "script_700_750.txt": creates plot in a new window

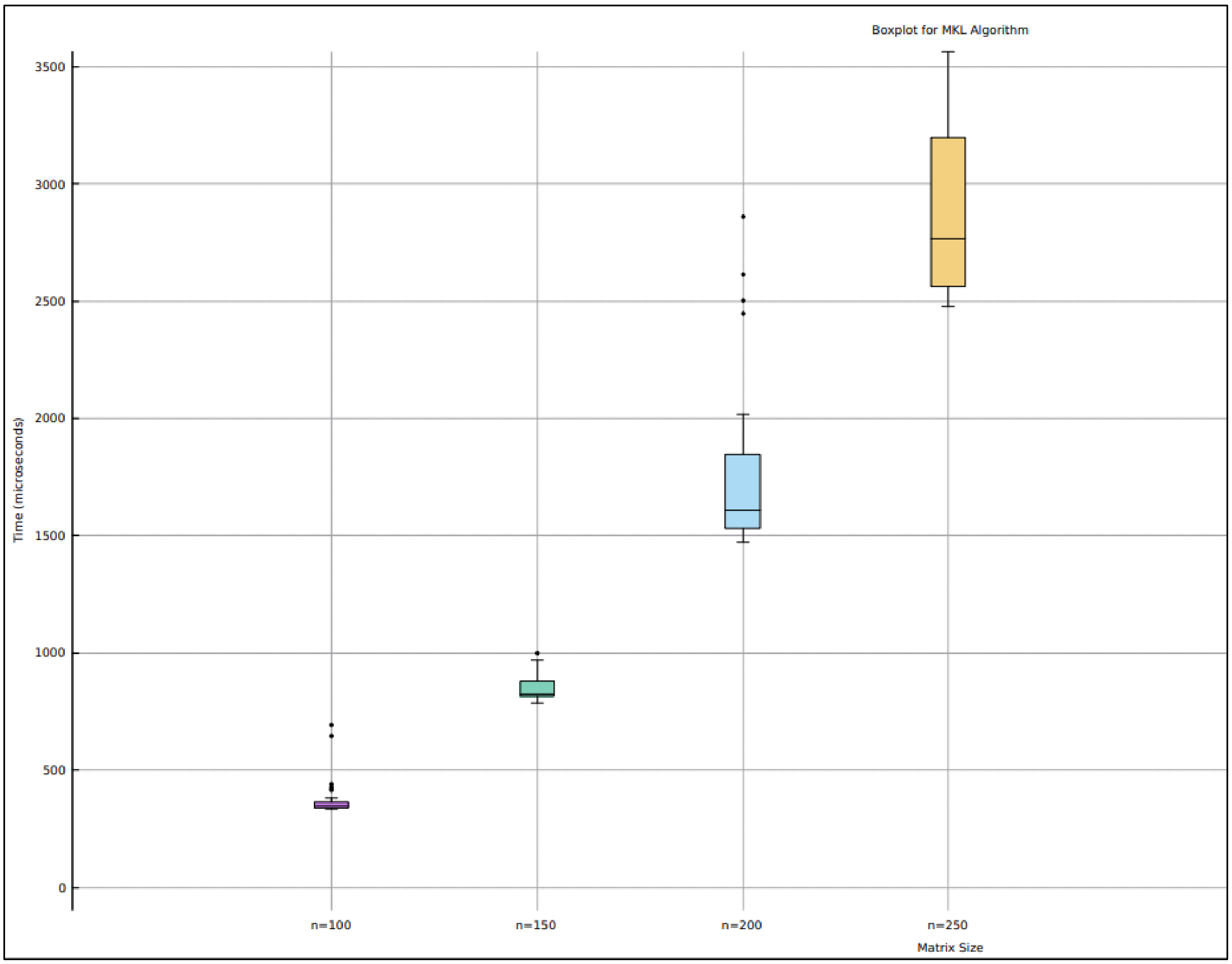
MKL

X axis displays the matrix size and y axis displays the time in microseconds



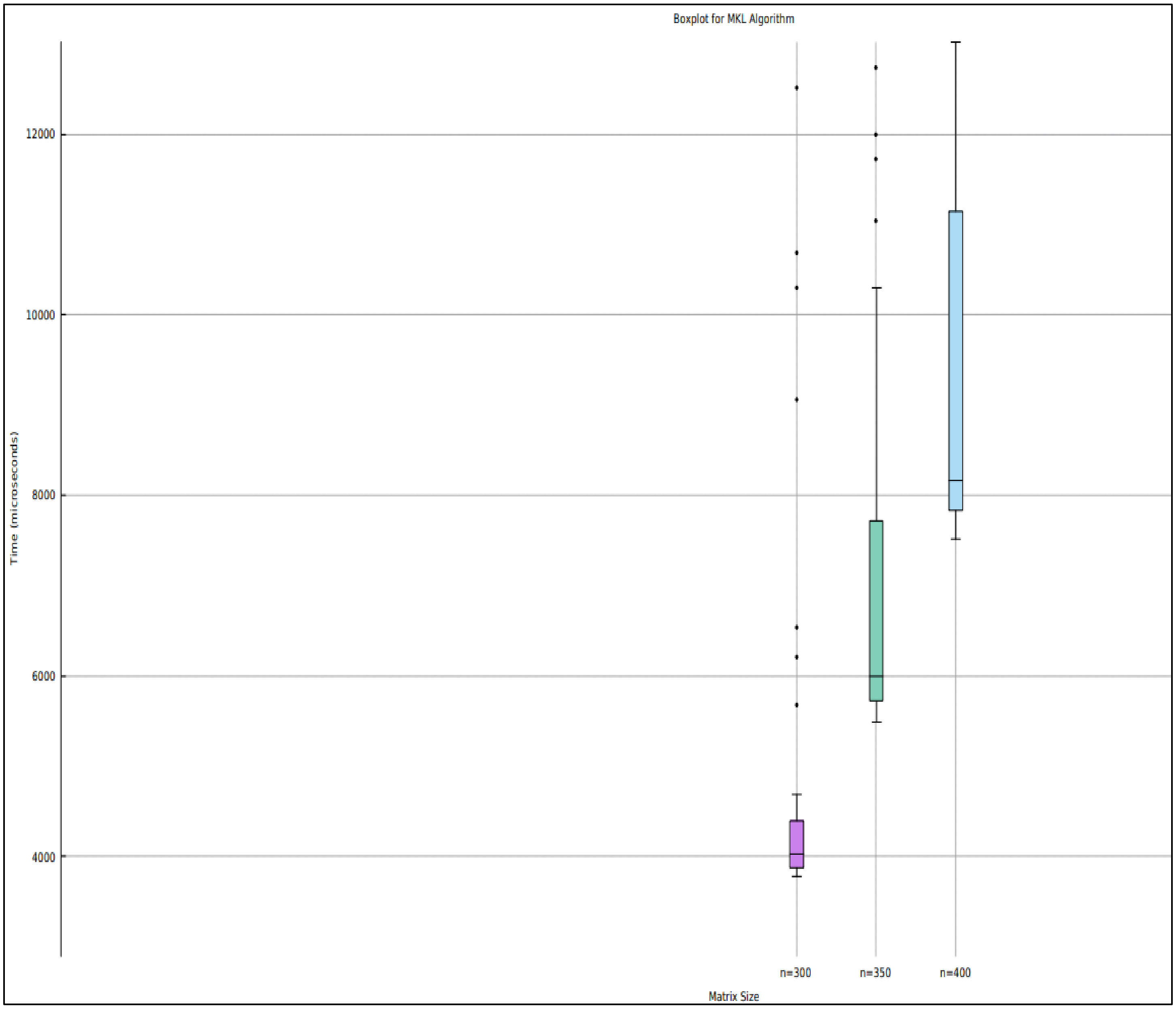
Plots for n=10 to n=90 (n=size of square matrix)

N	**10**	**30**	**60**	**80**
Mean	5.22	34.9	143.1	303.4
Standard Deviation	5.28	14.3	39.1	157.4



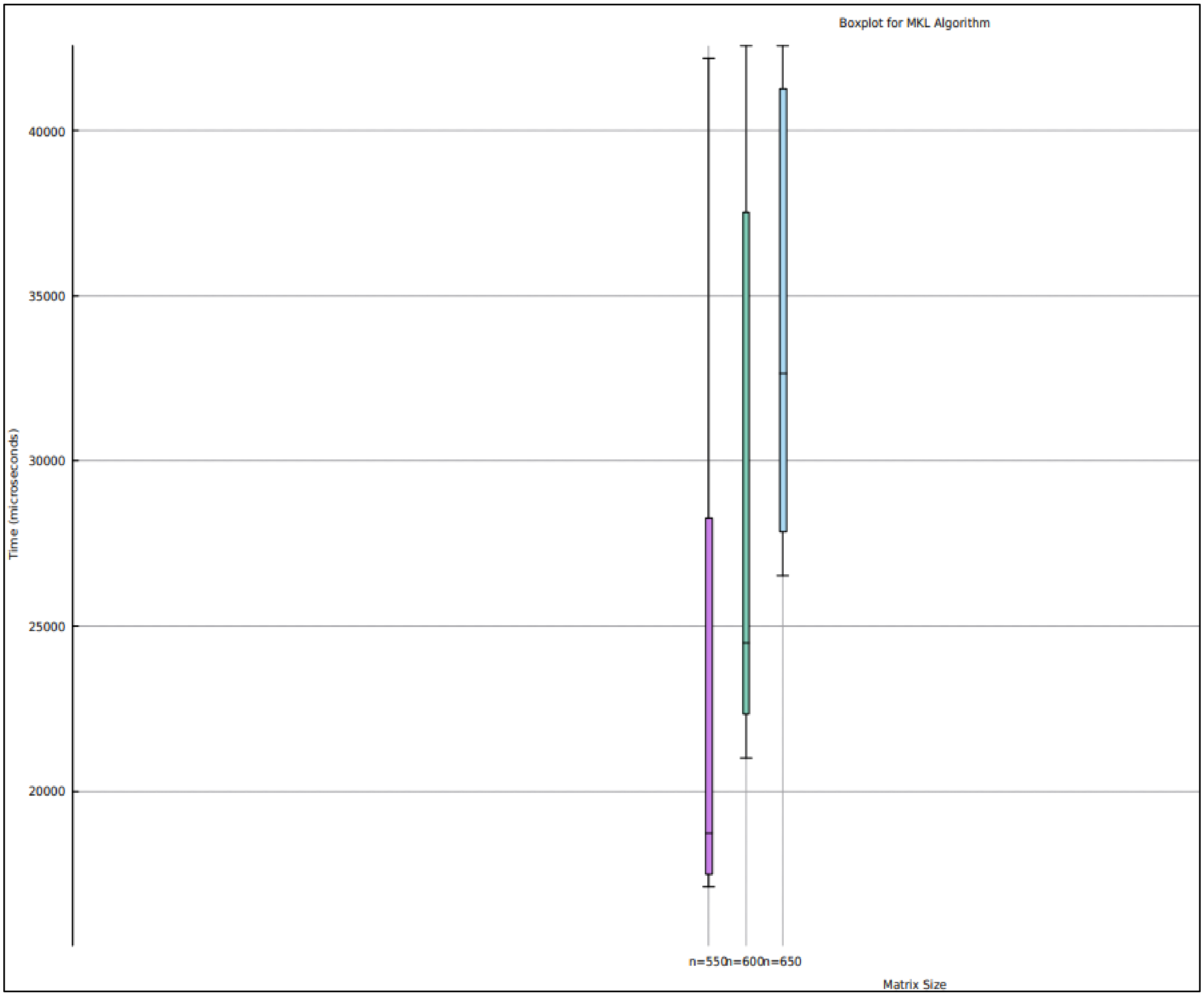
Plots for n=100 to n=250 (n=size of square matrix)

N	**100**	**200**	**250**
Mean	367.9	2088	3640
Standard Deviation	66	1245	2344



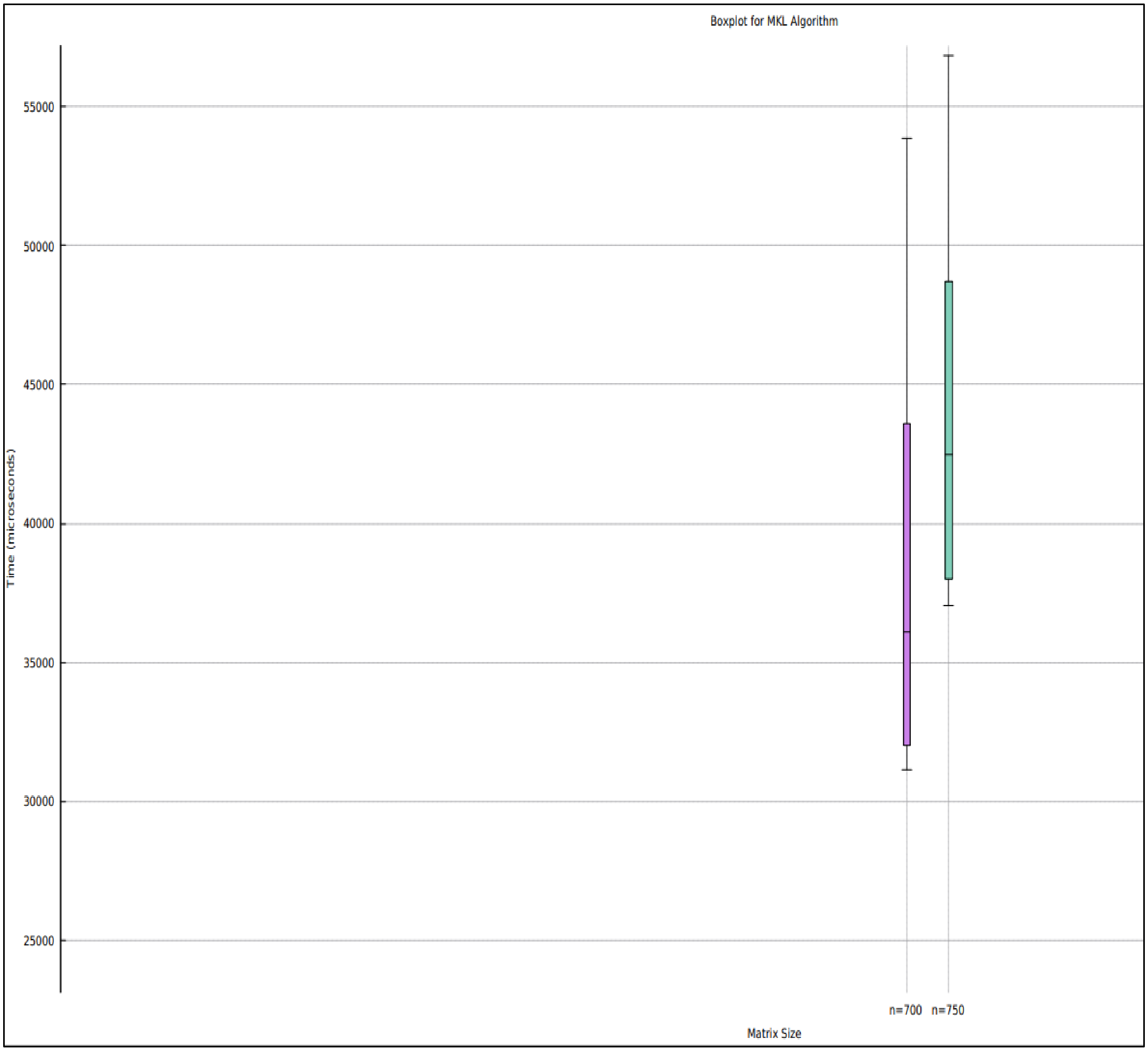
Plots for n=300 to n=400 (n=size of square matrix)

N	**300**	**400**
Mean	4875	10332
Standard Deviation	2220	4705



Plots for n=500 to n=650 (n=size of square matrix)

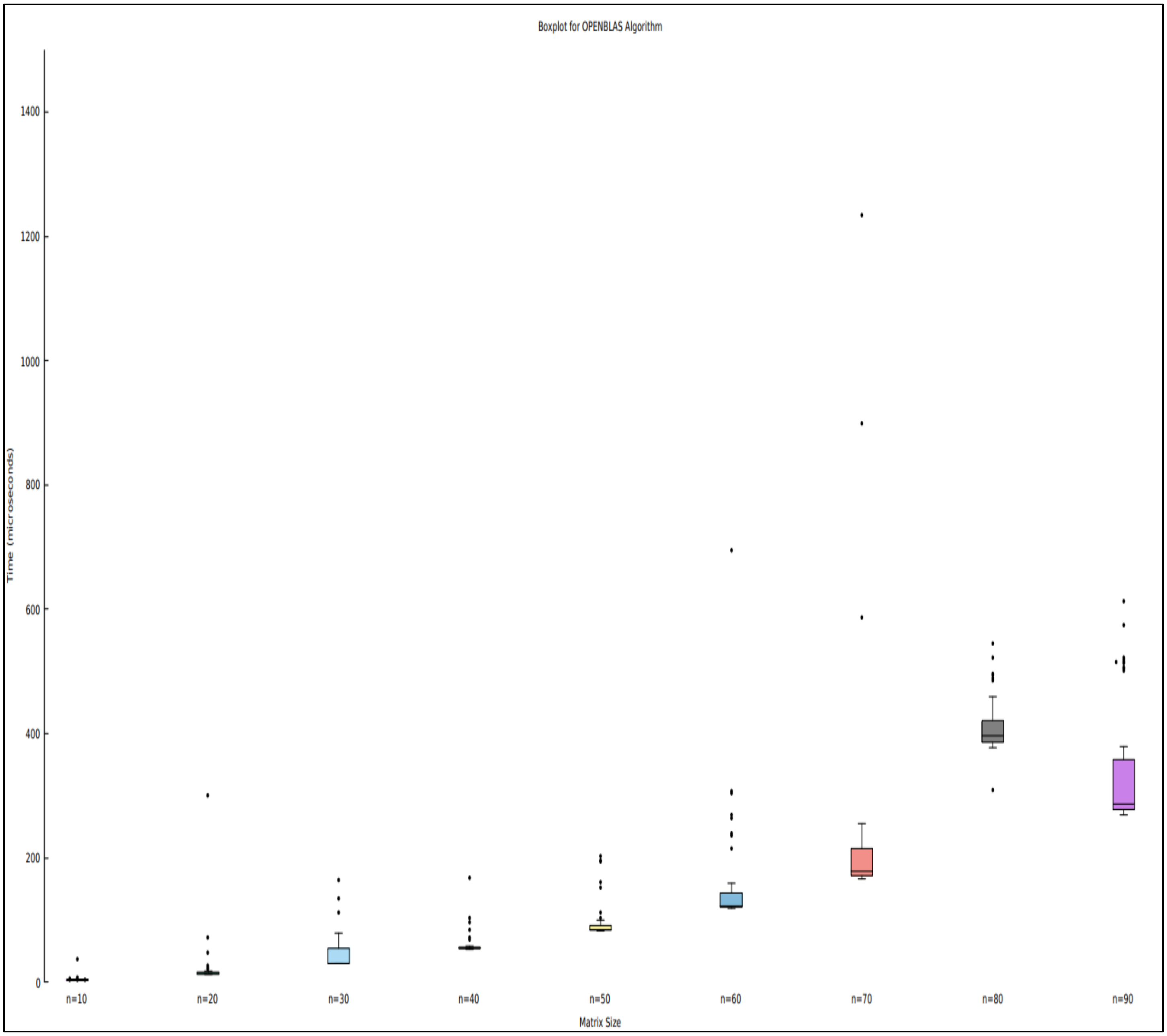
N	**550**	**650**
Mean	23772	35821
Standard Deviation	8859	8726



Plots for n=700 to n=750 (n=size of square matrix)

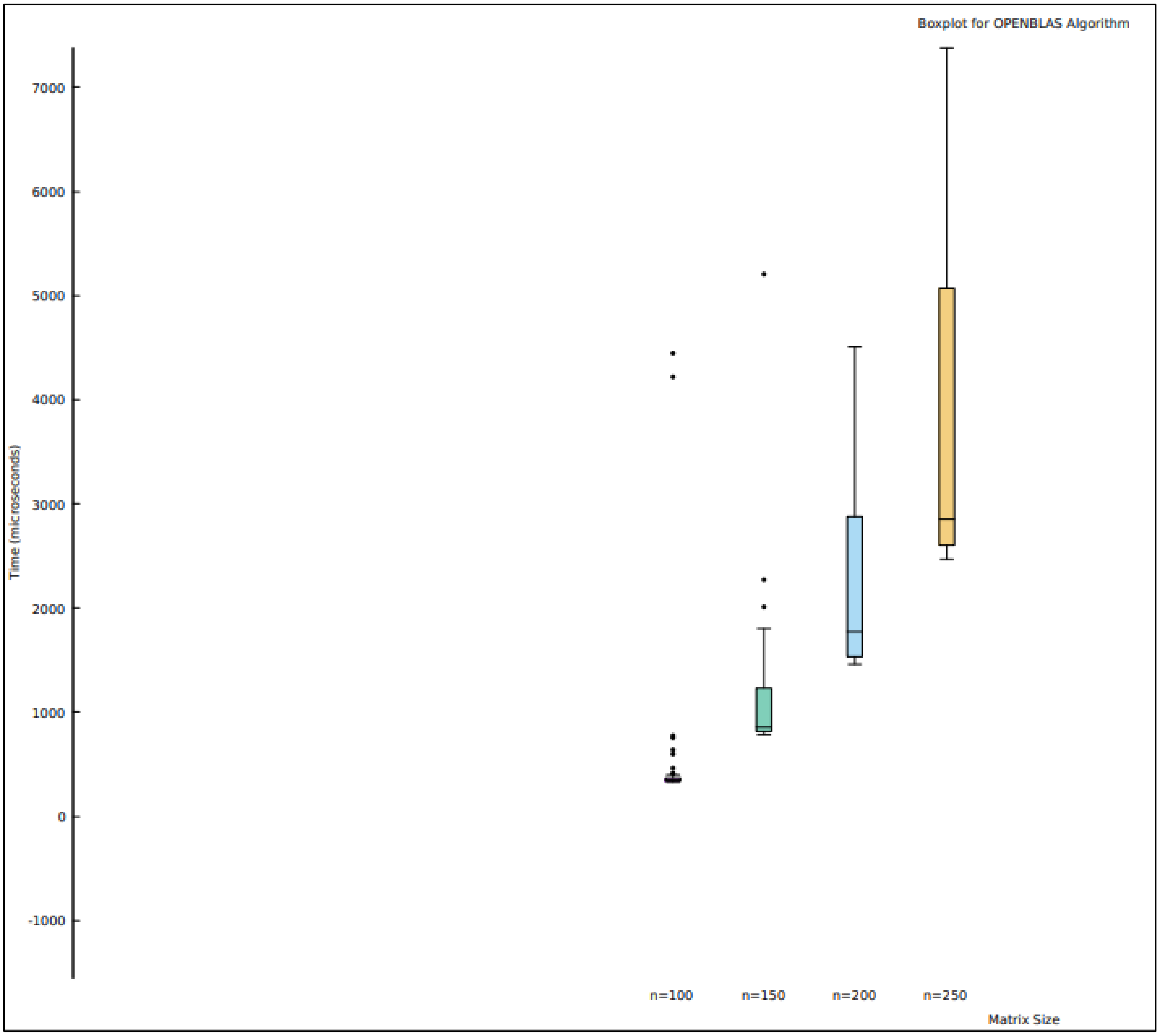
N	**700**	**750**
Mean	39150.8	43930.6
Standard Deviation	10422.8	5949.3

OpenBlas



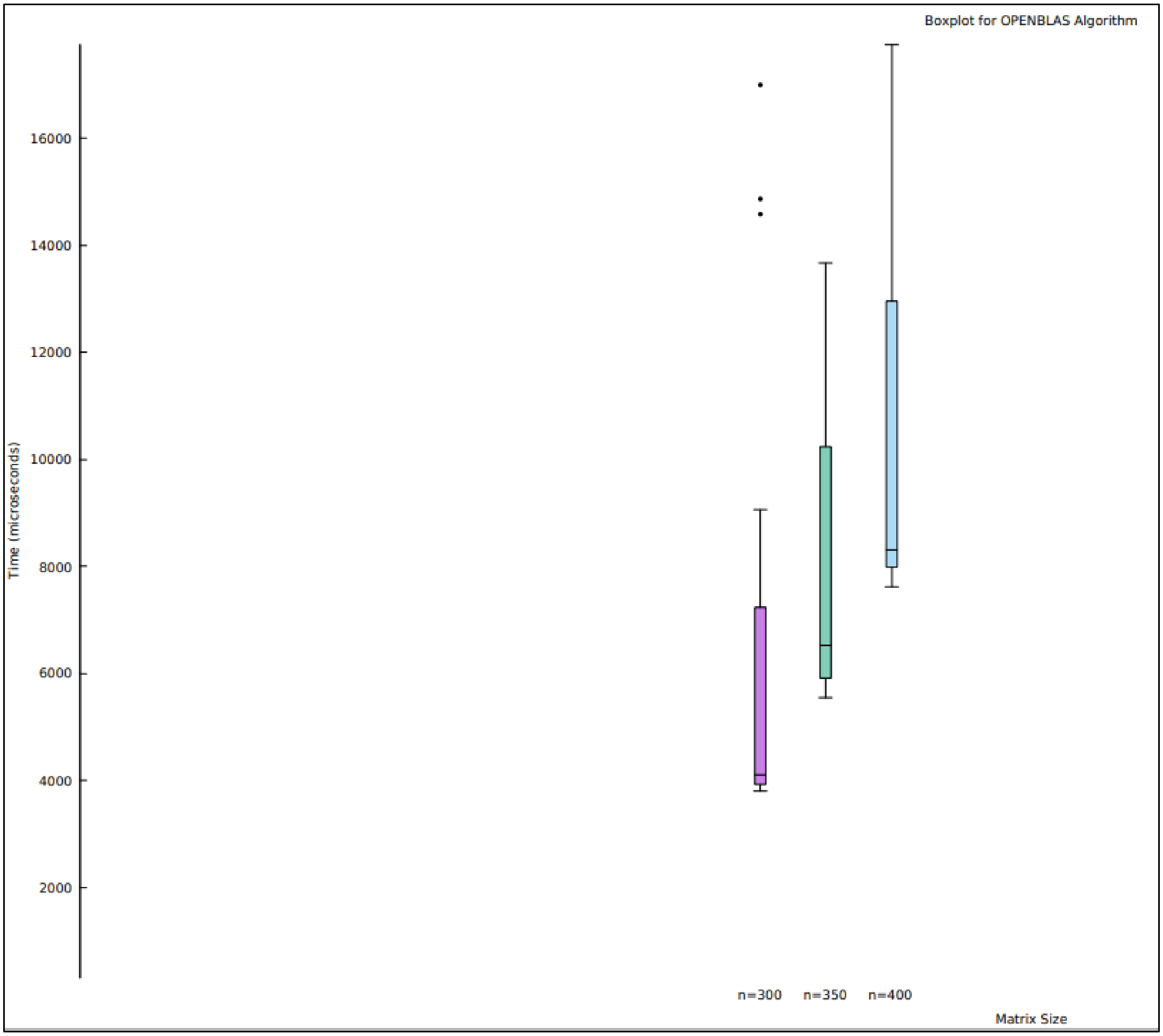
Plots for n=10 to n=90 (n=size of square matrix)

N	**10**	**30**	**60**	**80**
Mean	4.9	43.5	156.86	876.3
Standard Deviation	4.7	27.5	91.4	1956.9



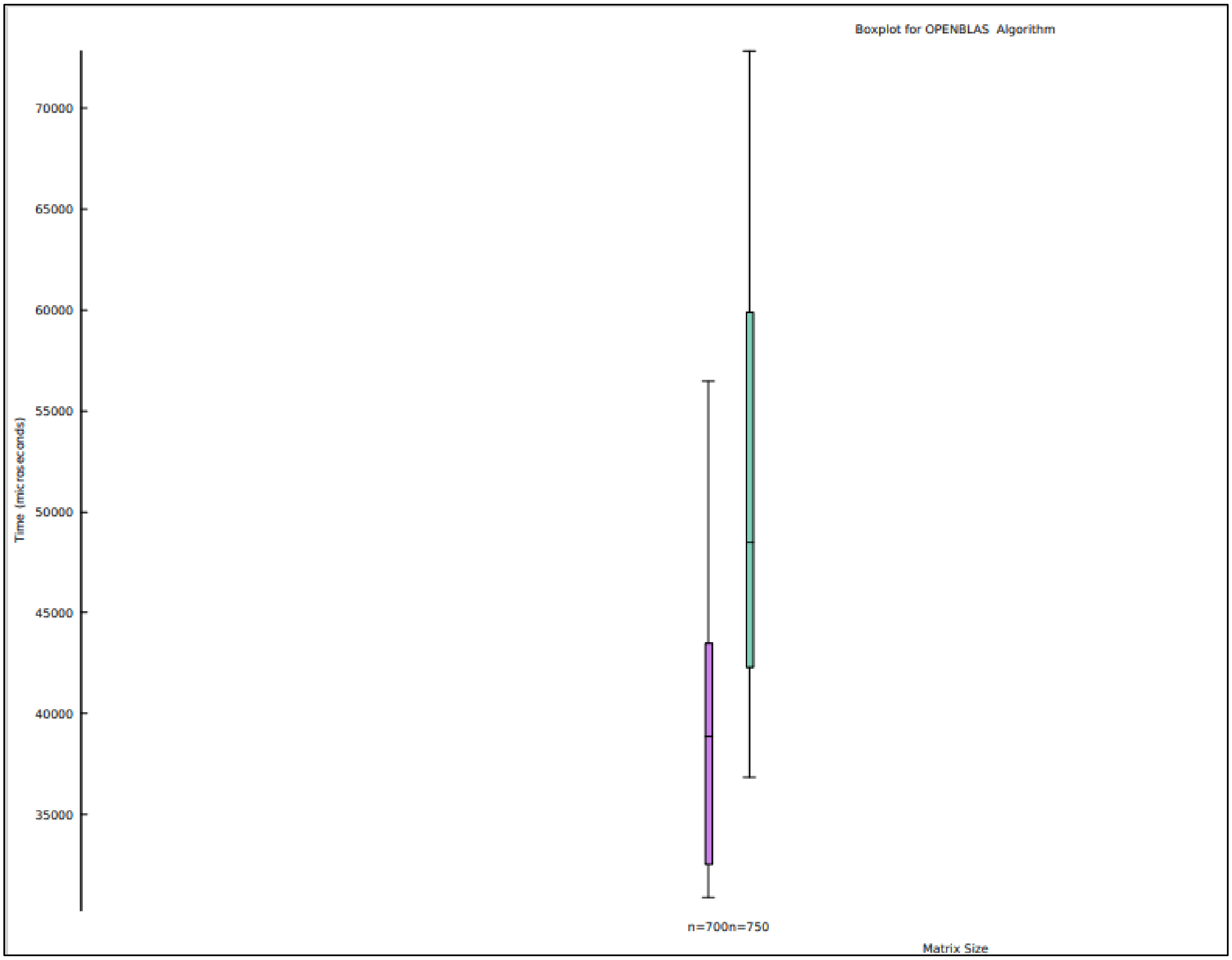
Plots for n=100 to n=250 (n=size of square matrix)

N	**100**	**200**	**250**
Mean	541.3	2805.4	4430.7
Standard Deviation	779.8	2473.8	3423.7



Plots for n=300 to n=400 (n=size of square matrix)

N	**300**	**400**
Mean	6649.8	11231.5
Standard Deviation	5037.3	5650.88

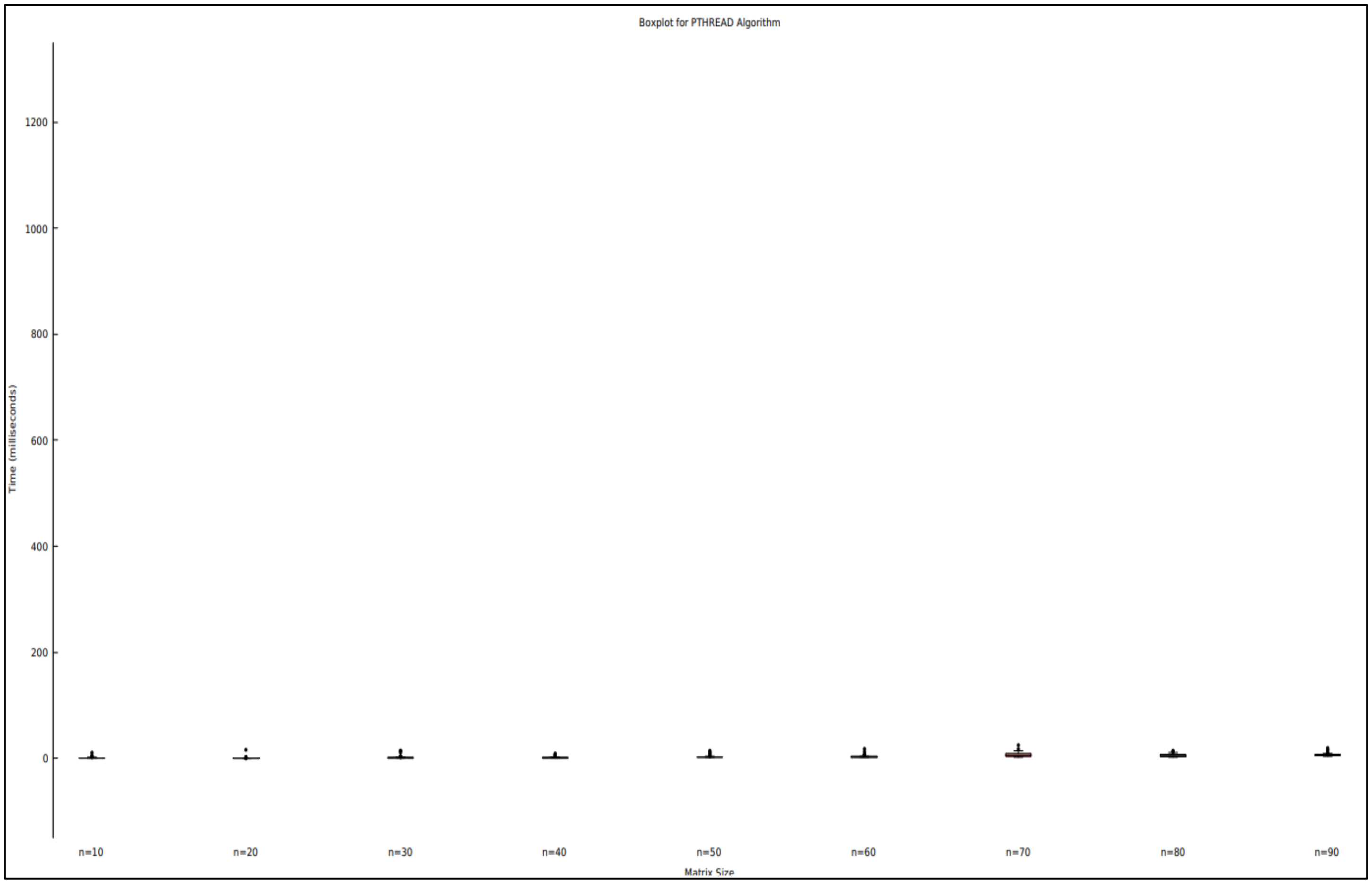


Plots for n=700 to n=750 (n=size of square matrix)

N	**700**	**750**
Mean	39939.4	50611.6
Standard Deviation	7440.41	10684.3

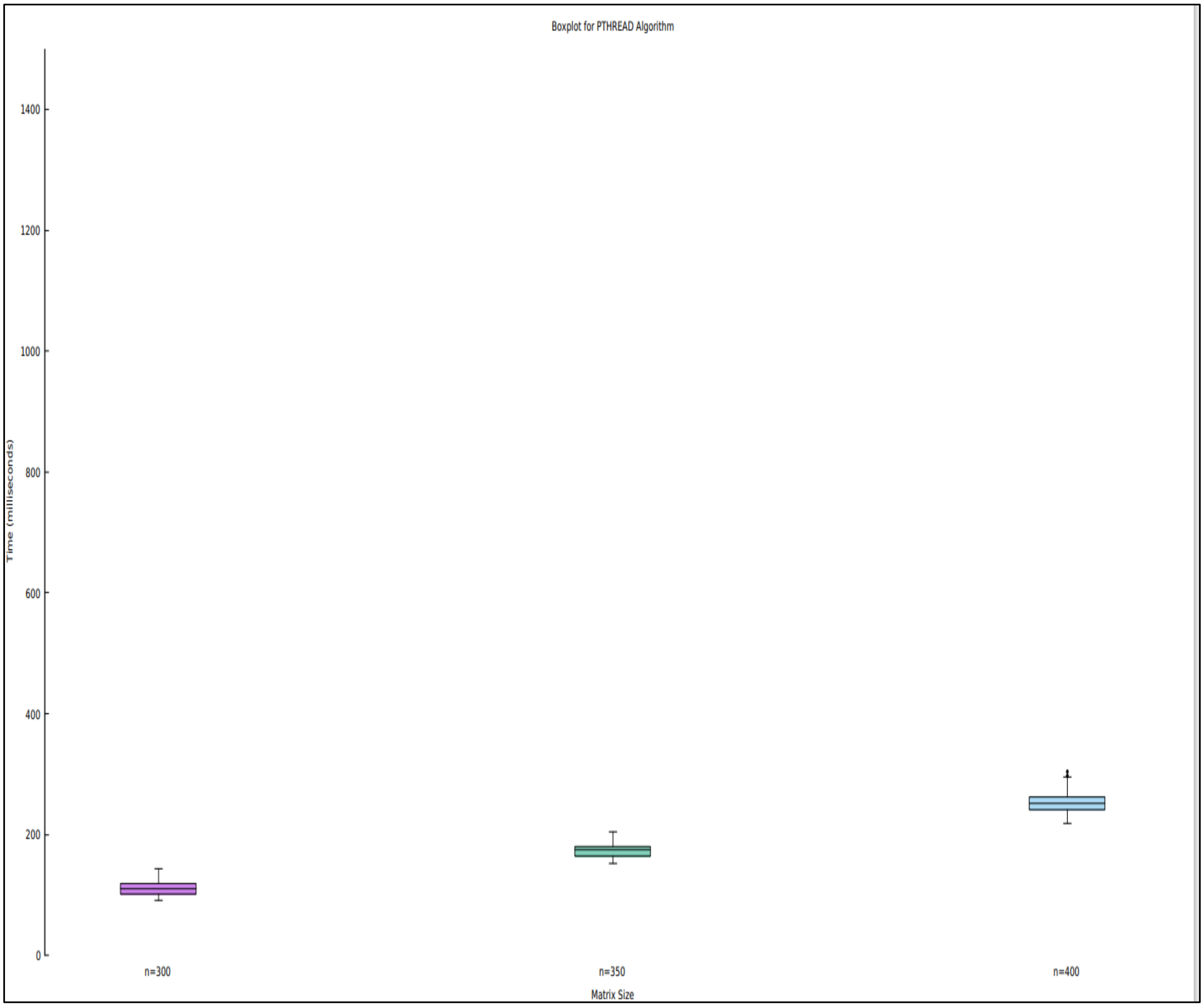
Pthread

Note: For pthread and naïve boxplots, time in milliseconds is used due to very large values.



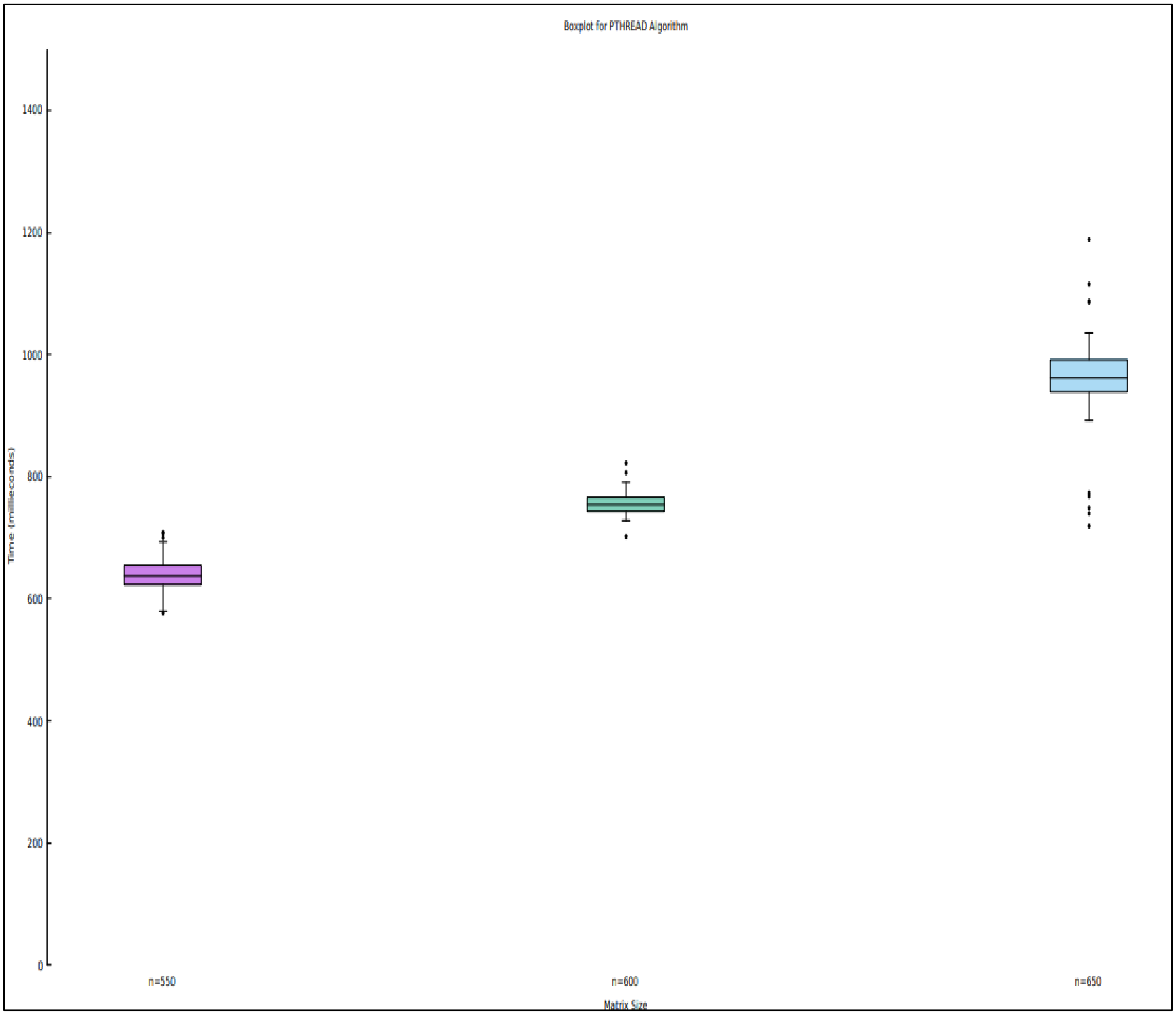
Plots for n=10 to n=90 (n=size of square matrix)

N	**10**	**30**	**60**	**80**
Mean (microseconds)	1610.96	2312.34	3784.7	6137.36
Standard Deviation(microseconds)	1760.31	3086.13	2815.04	2804.51



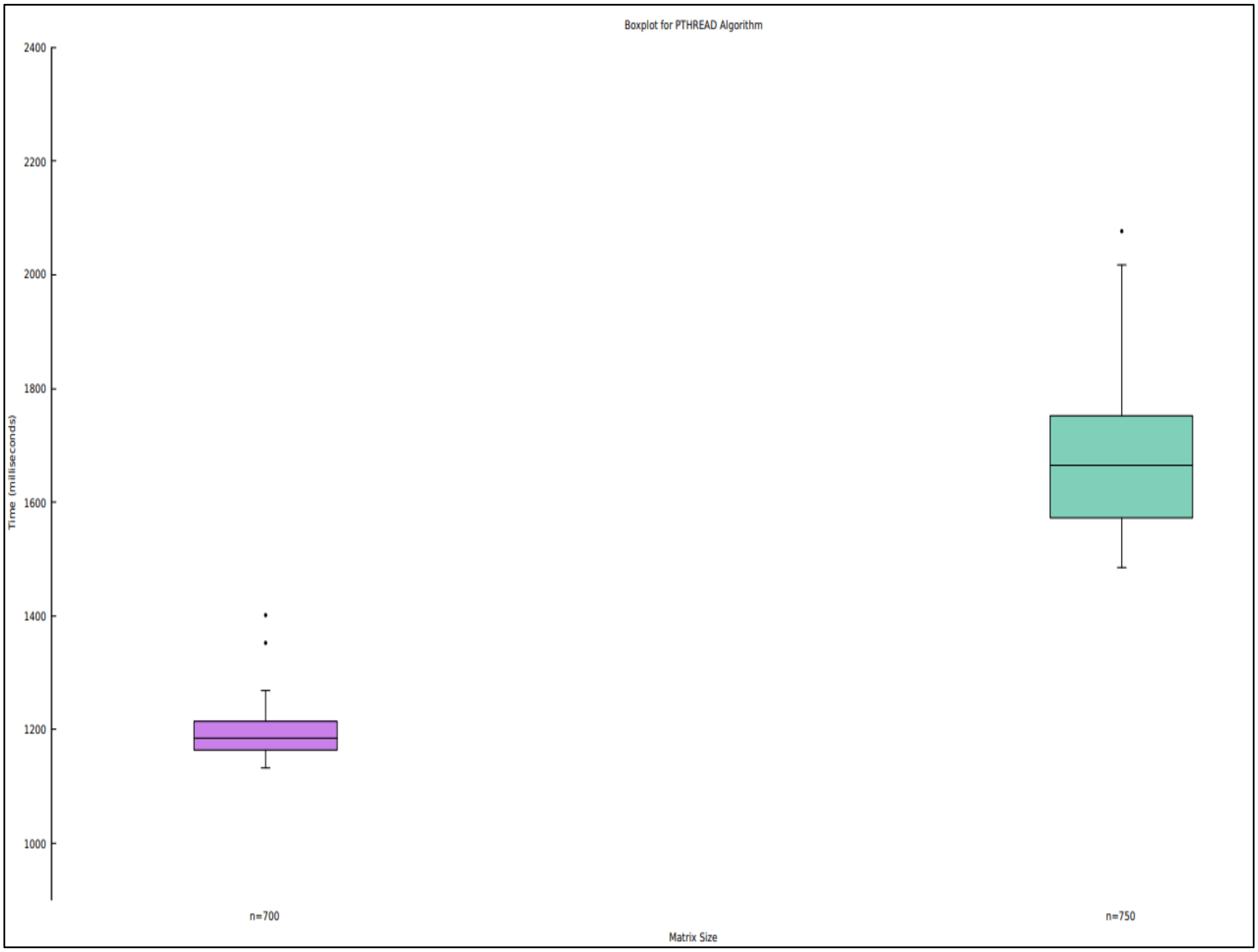
Plots for n=300 to n=400 (n=size of square matrix)

N	**300**	**400**
Mean	111194	253677
Standard Deviation	11000	17769.8



Plots for n=550 to n=650 (n=size of square matrix)

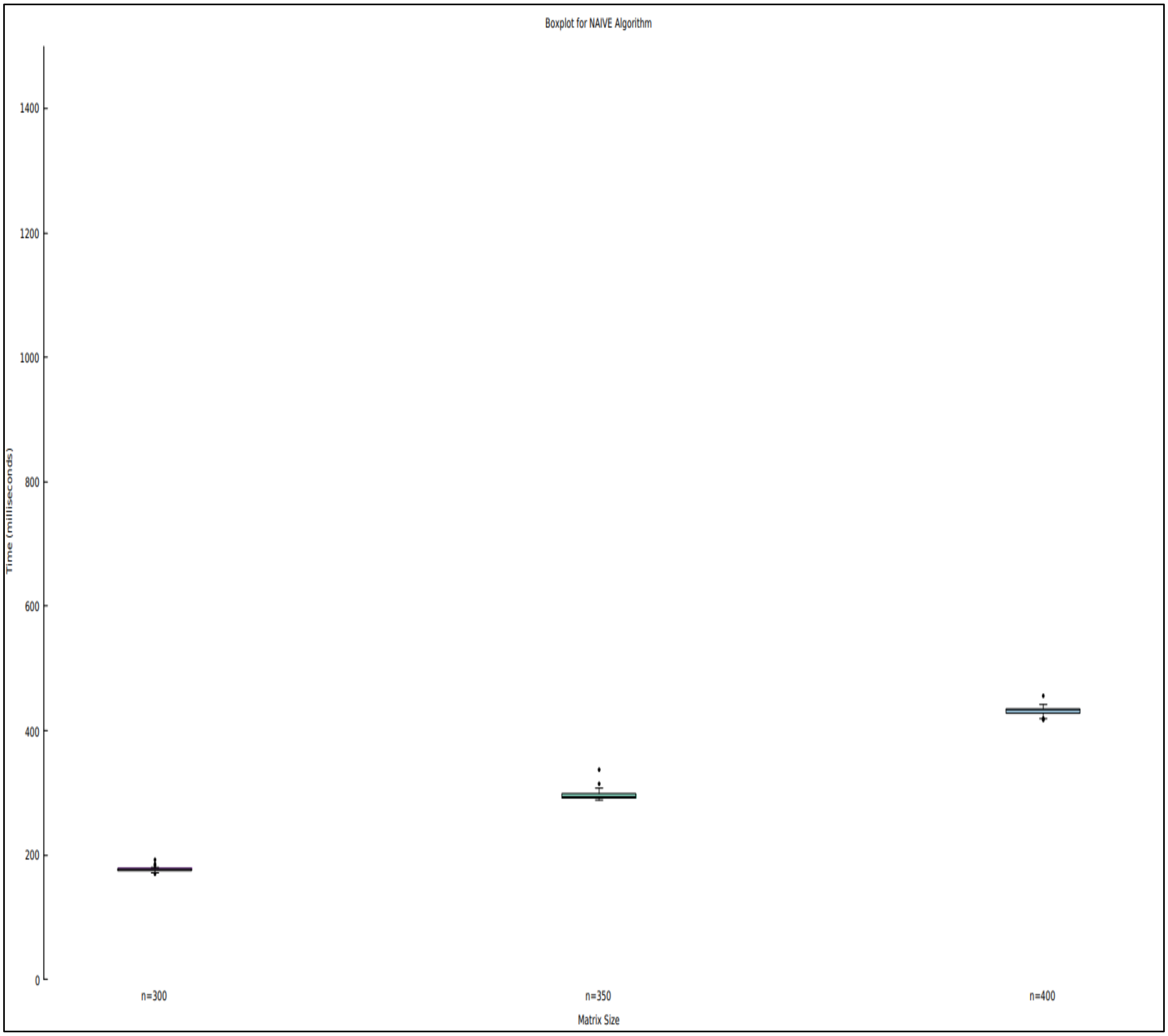
N	**550**	**650**
Mean	637448	955183
Standard Deviation	31020	85292



Plots for n=700 to n=750 (n=size of square matrix)

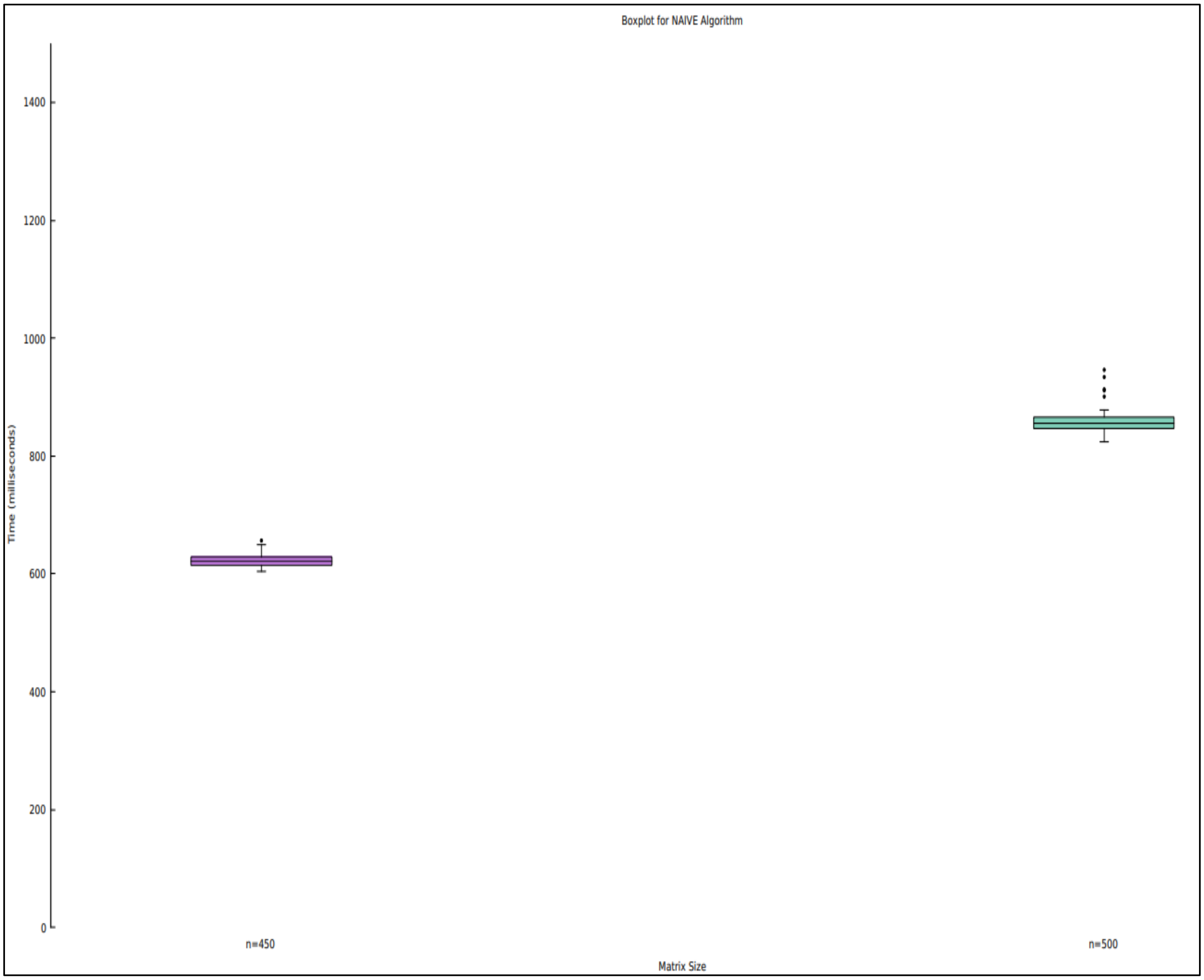
N	**700**	**750**
Mean	1.19e+06	1.68e+06
Standard Deviation	49988	145564

Naïve



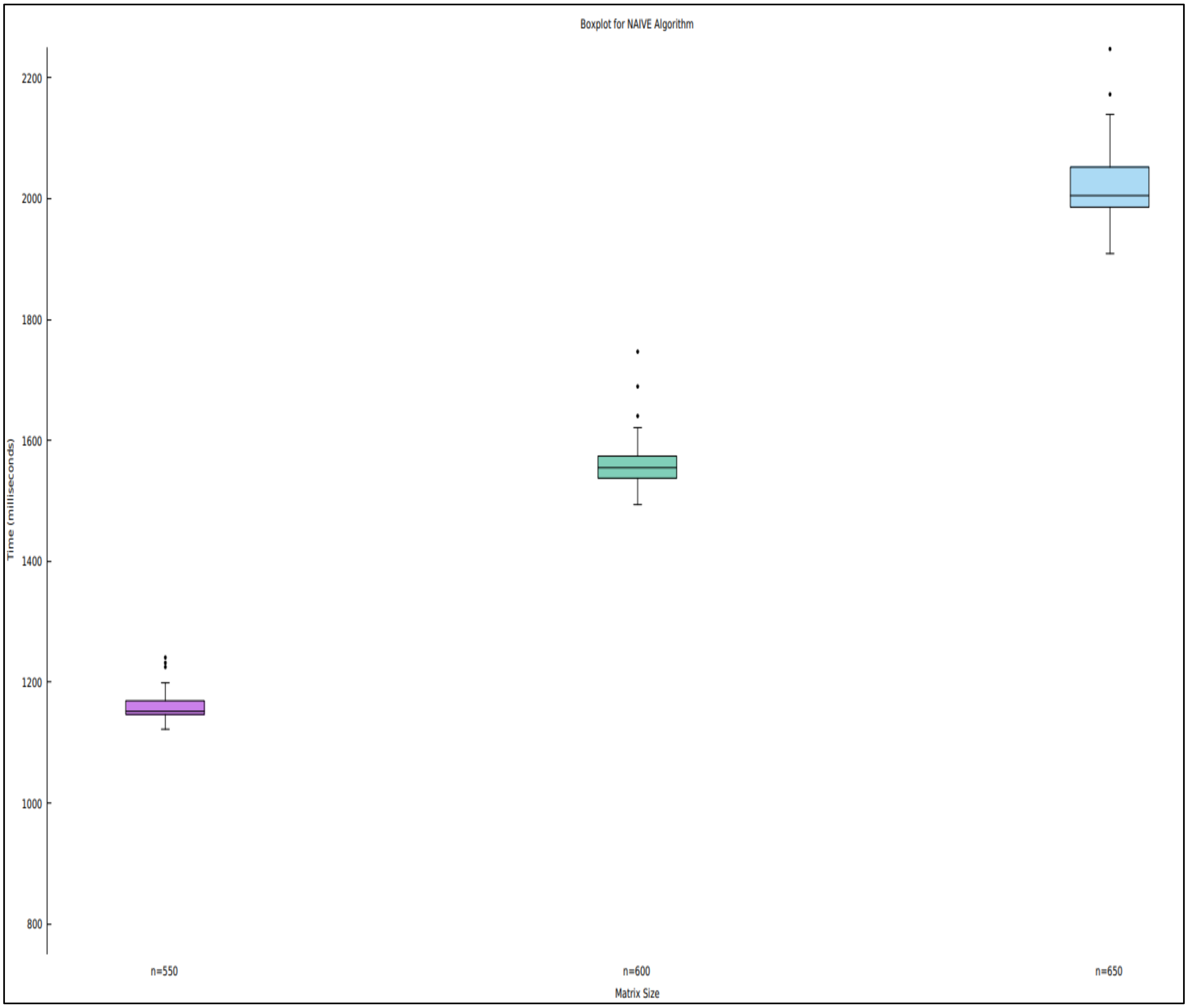
Plots for n=300 to n=400 (n=size of square matrix)

N	**300**	**400**
Mean	177396	431642
Standard Deviation	3727.37	6981.39



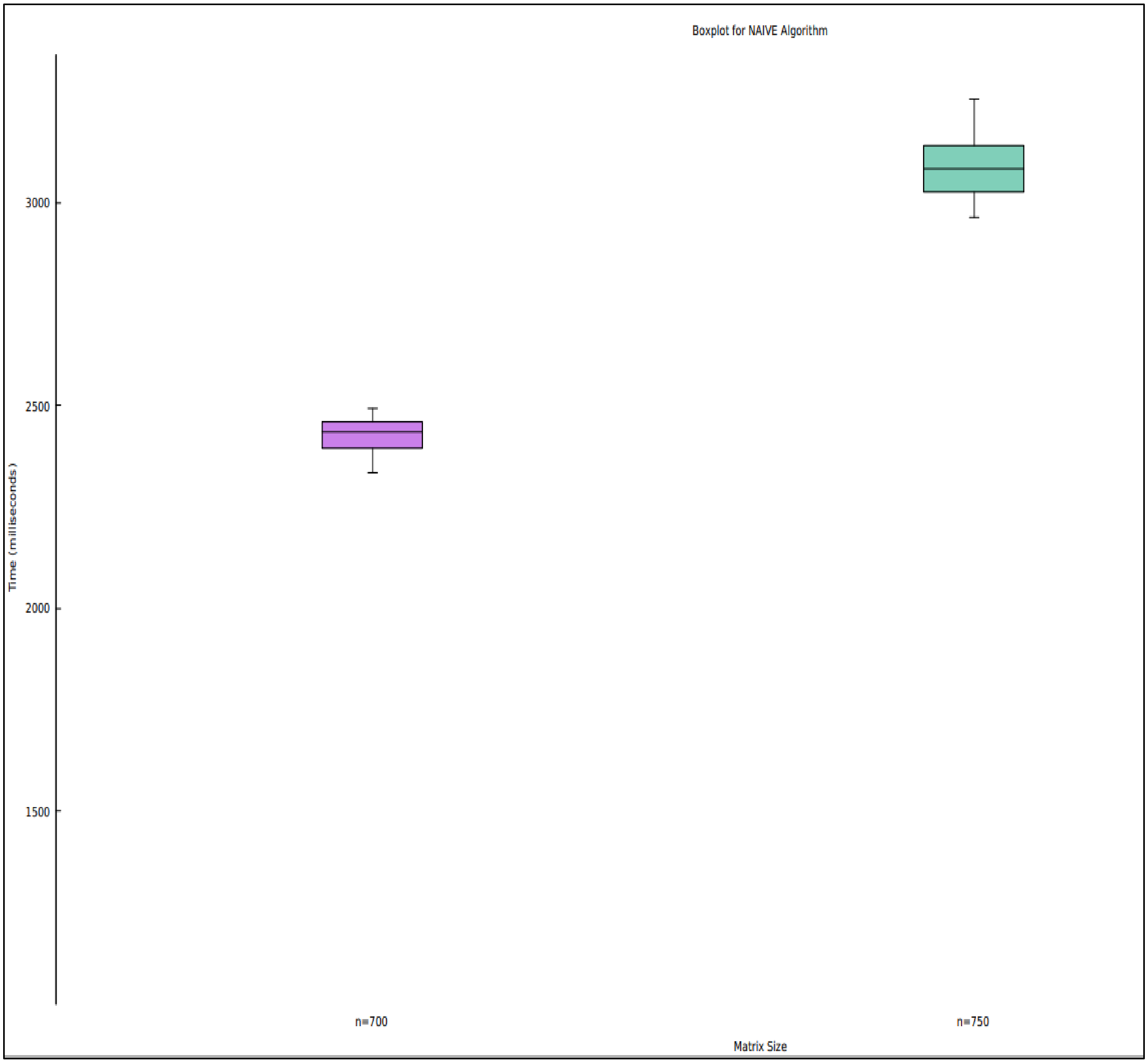
Plots for n=450 to n=500 (n=size of square matrix)

N	**450**	**500**
Mean	623157	860593
Standard Deviation	11475.2	24390.7



Plots for n=550 to n=650 (n=size of square matrix)

N	**550**	**650**
Mean	1.16e+06	2.03e+06
Standard Deviation	25899	93674.1



Plots for n=700 to n=750 (n=size of square matrix)

N	**700**	**750**
Mean	2.42e+06	3.09e+06
Standard Deviation	46734.2	72415.2