Assignment 2

Parallel Matrix Multiplication

433-678 - Cluster and Grid Computing

Scott Beca (sbeca) – 267 362

s.beca@ugrad.unimelb.edu.au

Introduction

We were tasked with creating a program that computed the result of a matrix multiplication of two square matrices using parallel programming. To implement the parallel programming side, we were asked to use an API called Message Passing Interface (MPI). We had to create matrices of different sizes with randomly generated elements, split up the work between different numbers of worker nodes and prove that using more nodes would speed up the computation time.

Implementation

For our implementation, we decided that we would share the work load between worker processes by assigning them a number of rows in the final matrix to compute. The number of rows each process had to compute is calculated with:

$$Rows = \frac{Matrix\ Size}{Number\ Of\ Worker\ Processes}$$

Any remainder rows are assigned individually to individual worker processes.

This required us to send each worker process the entire second matrix and the rows in the first matrix that correspond to the rows they had been assigned to compute in the final matrix. With this data, each worker process was able to compute their assigned rows in parallel.

Problems

The University of Melbourne Computer Science department supplied the students of this class a cluster set-up for use for this assignment. However, this cluster reportedly suffered from many different problems during the time we were given to work on this assignment. When the time came for us to compile and test our program on this cluster, our program would not compile even though we were able to compile it without error on our home test computers.

Other students mentioned having problems compiling on the subject's Discussion Board so we decided to use our home computer to do the testing.

Just before our demonstration on Thursday 28th of April, we were able to find and fix the problems with our program. We will leave the information and results that relate to using our home computer and also add our new results to this second report submission.

Test Set Up – Home Computer

Our test computer runs a quad-core CPU, has 8GB of RAM and runs Windows 7 64-bit. The MPI software we used is called MPICH2.

We tested with 5 different matrix sizes: 100, 200, 300, 400 and 500.

We tested with 4 different numbers of processes: 2, 3, 4 and 8.

To get a good spread of data, we did 20 runs for each of the different settings, resulting in 400 sets of data.

Test Set Up – Manjra Cluster

We tested with 5 different matrix sizes: 100, 200, 300, 400 and 500.

We tested with 4 different numbers of processes: 2, 4, 6 and 8.

To get a good spread of data, we did 10 runs for each of the different settings, resulting in 200 sets of data.

Results – Home Computer

The results are included after this page.

The attached table includes the averages of each of the 20 runs (excluding the shortest and longest times, to remove outliers).

Graphs are included for each of the matrix sizes (one with all the data and one with just the averages).

Because our test computer includes 4 cores, we expected to see an increase in the speed of computation up to 4 processes as these processes could run in parallel and do more work in a shorter time. As we increase the number of processes past 4, we expected performance to decrease as we can't do any more work in parallel and we are adding more overhead in the messaging and running of more processes.

The results we got reflect what we expected.

Results – Manjra Cluster

The results are included after this page.

The attached table includes the averages of each of the 10 runs.

Graphs are included for each of the matrix sizes (one with all the data and one with just the averages).

Because the Manjra Cluster had (at the time of running our tests) 8 available nodes for use, we expected to see an increase in the speed of computation up to 8 processes as these processes could run in parallel and do more work in a shorter time.

The results we got reflect what we expected to the most part. When computing a matrix of size 100, it took longer to compute the results when using 8 nodes over 6 nodes. As a matrix of size 100 is the smallest size we tested, this can be explained by contributing that decrease in performance to more overhead in creating and controlling more processes and having more message sending to deal with.

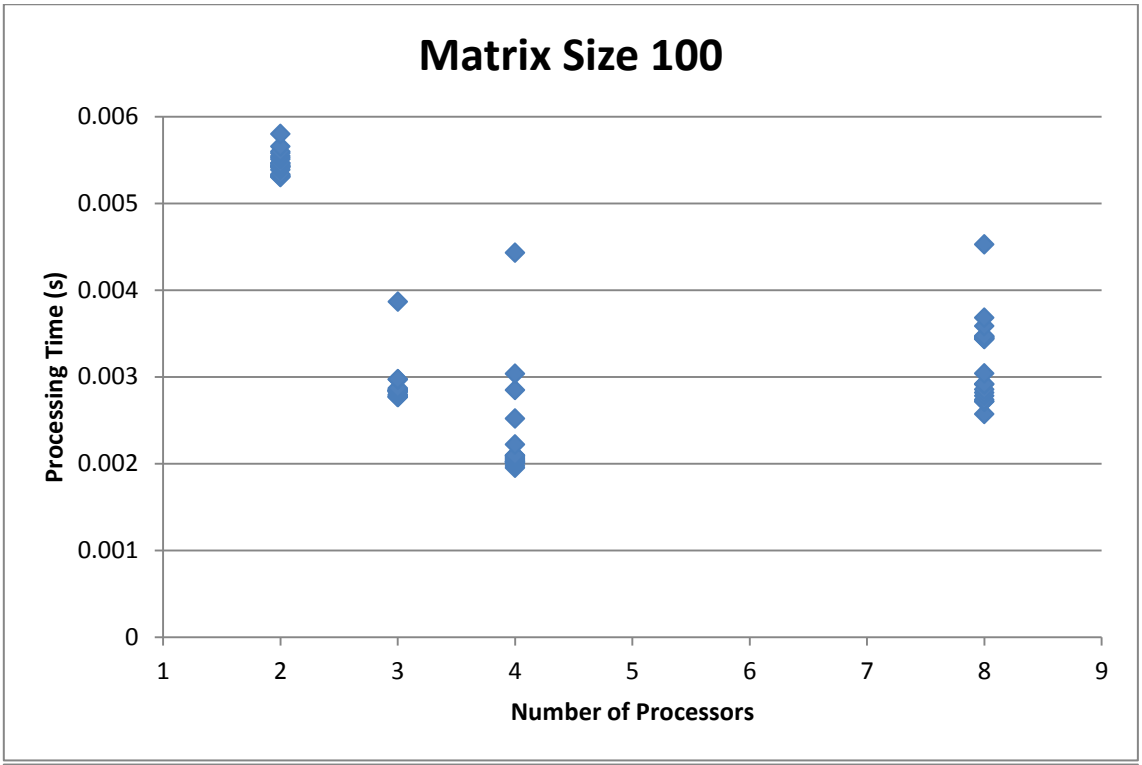
Result Averages on Home Computer:

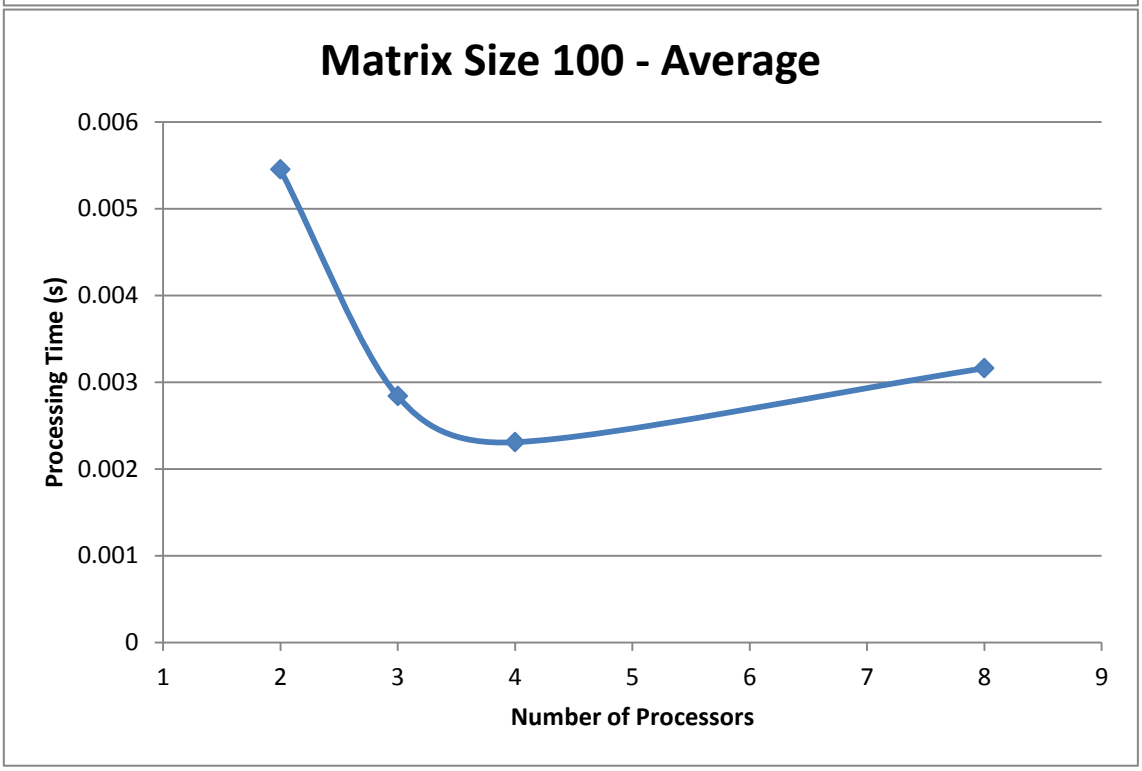
Matrix Size	Number of Processors	Average Processing Time (s)
100	2	0.005449833
100	3	0.002837944
100	4	0.002307778
100	8	0.003160611
200	2	0.043000444
200	3	0.022300944
200	4	0.016005389
200	8	0.100558111
300	2	0.148017889
300	3	0.076670556
300	4	0.060287722
300	8	0.160325111
400	2	0.356953056
400	3	0.183539500
400	4	0.135044056
400	8	0.716865722
500	2	0.711932056
500	3	0.370395944
500	4	0.280381889
500	8	0.900066111

Result Averages on Manjra Cluster:

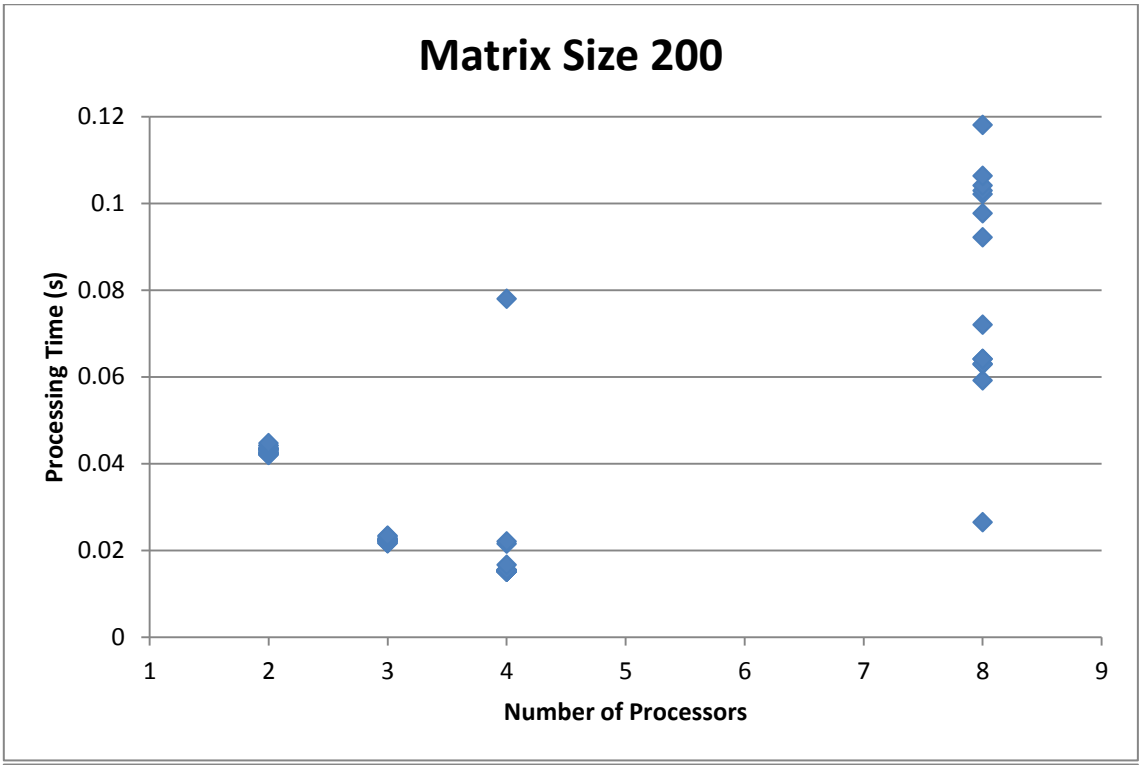
Matrix Size	Number of Processors	Average Processing Time (s)
100	2	0.024885667
100	4	0.012380222
100	6	0.010483444
100	8	0.012623556
200	2	0.167830222
200	4	0.069144111
200	6	0.054717444
200	8	0.040853667
300	2	0.715655667
300	4	0.221554667
300	6	0.148685444
300	8	0.121334889
400	2	1.644523444
400	4	0.573082333
400	6	0.387347333
400	8	0.297735111
500	2	4.124398444
500	4	1.228933444
500	6	0.741369556
500	8	0.547622222

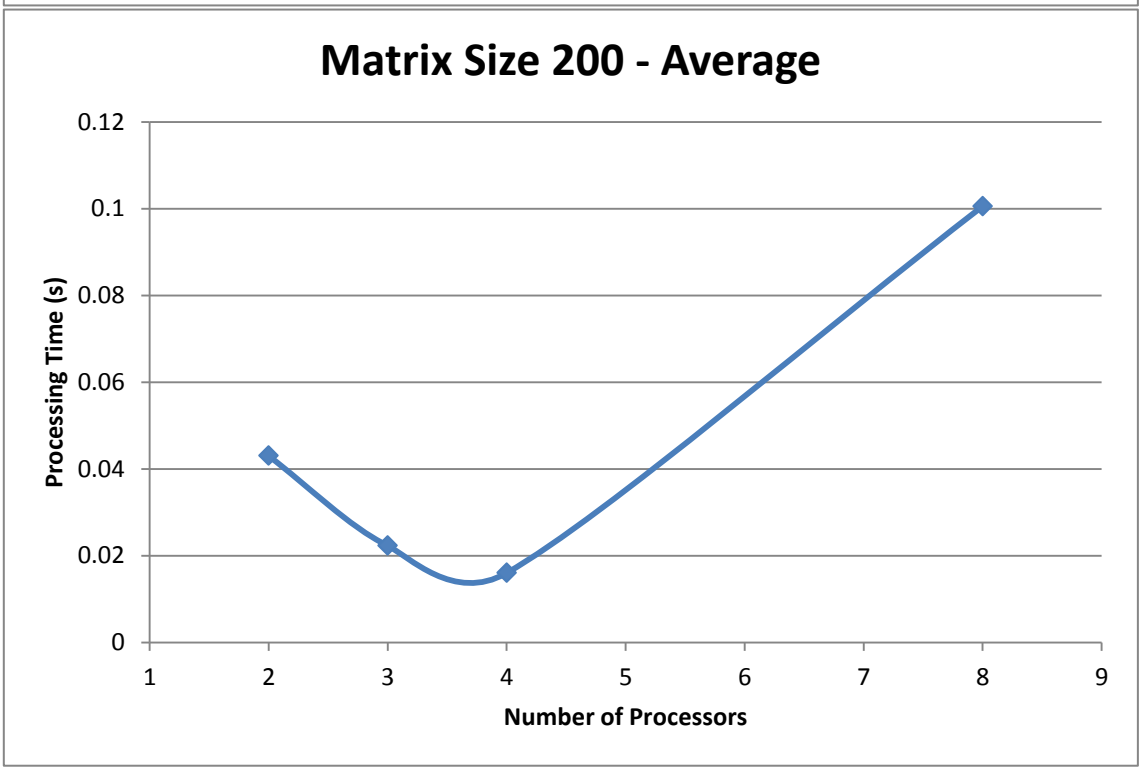
Home Computer Results:



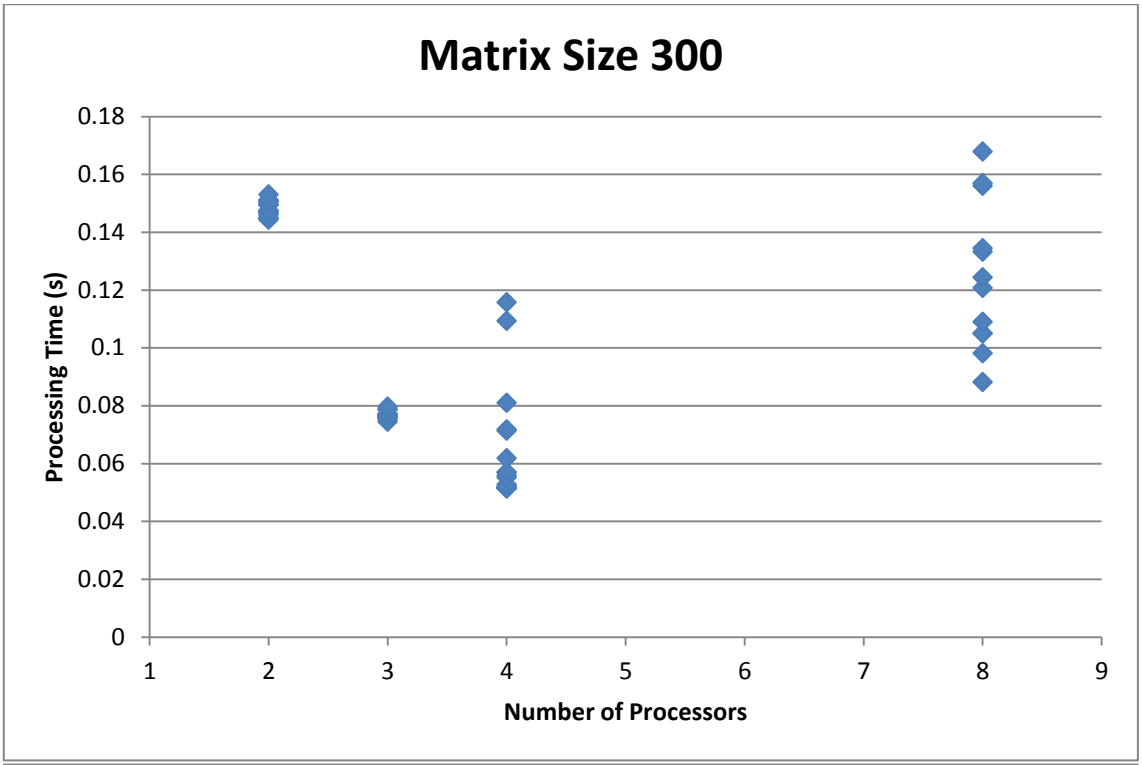


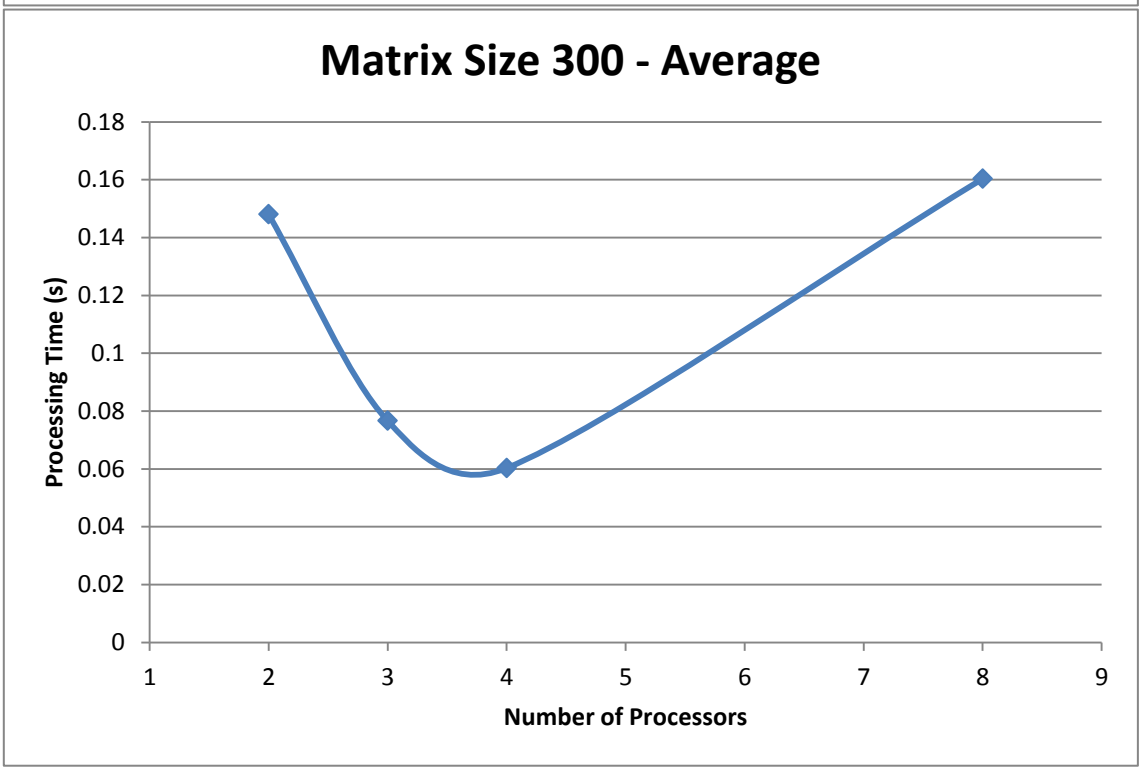
Home Computer Results:



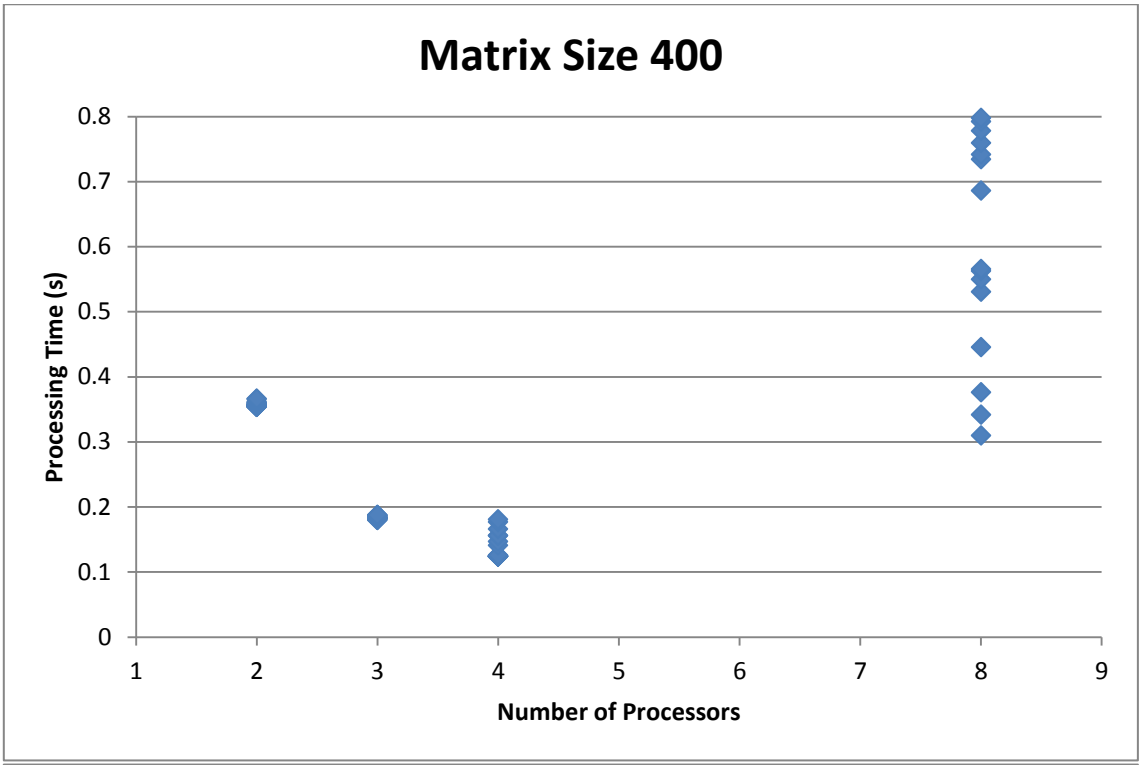


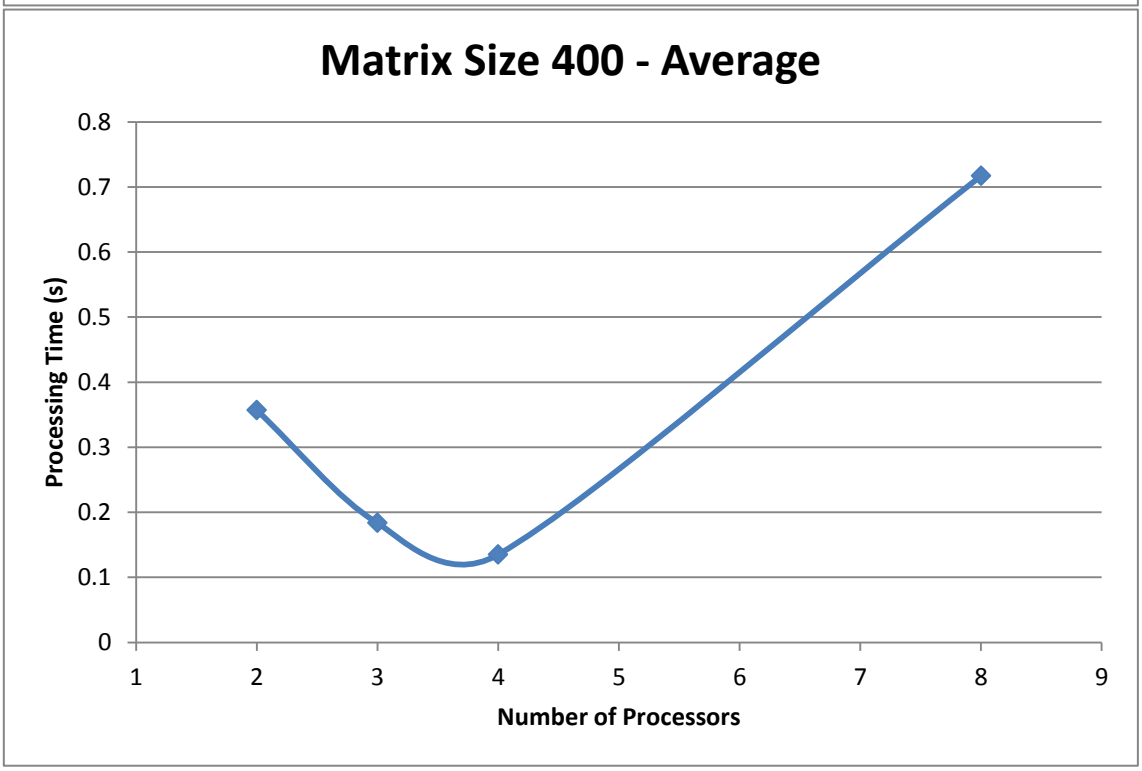
Home Computer Results:



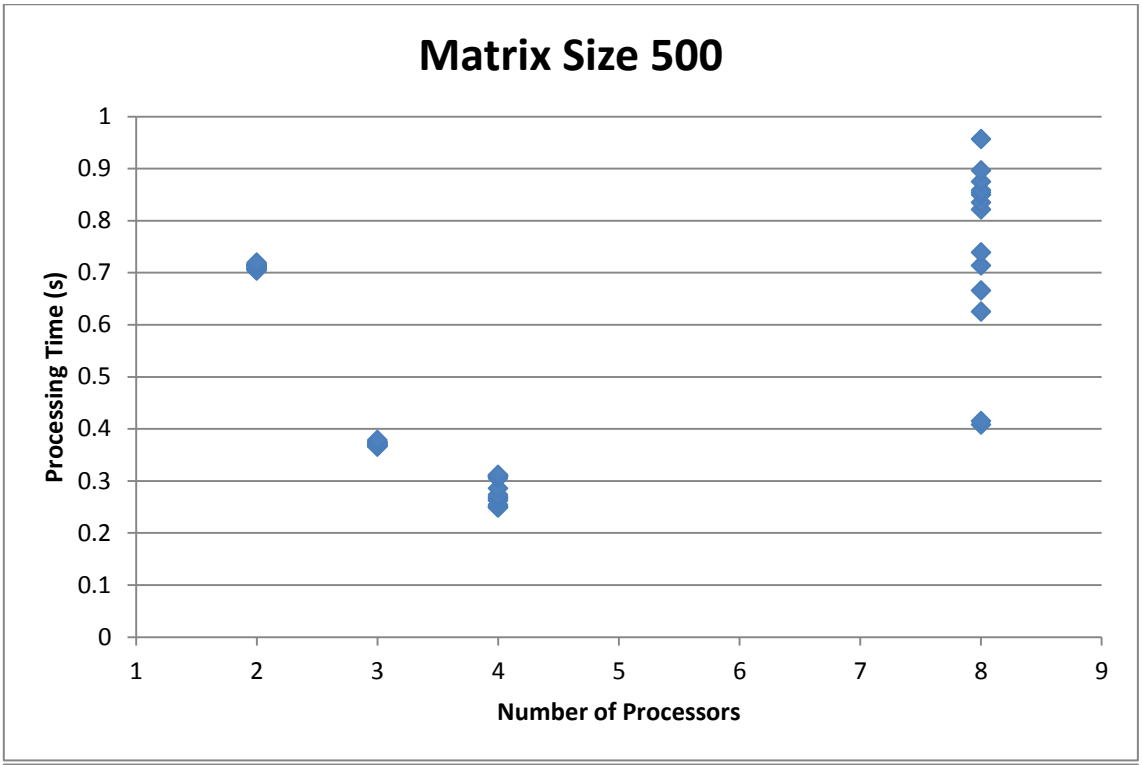


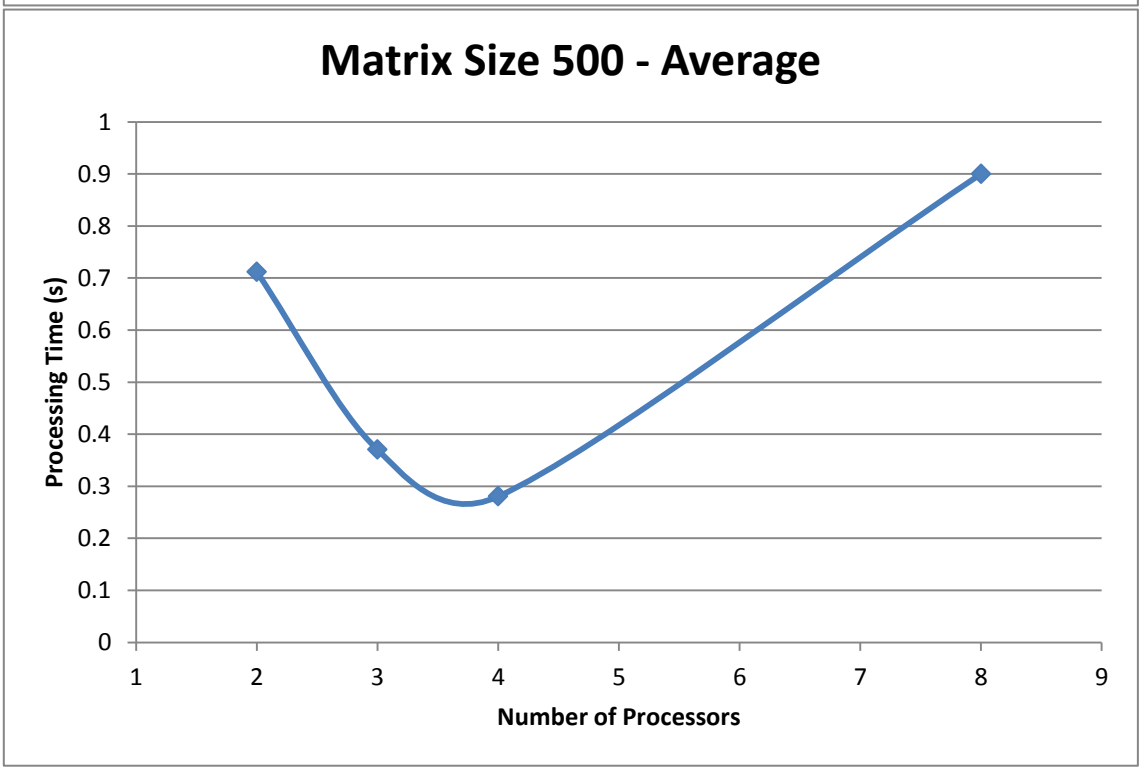
Home Computer Results:



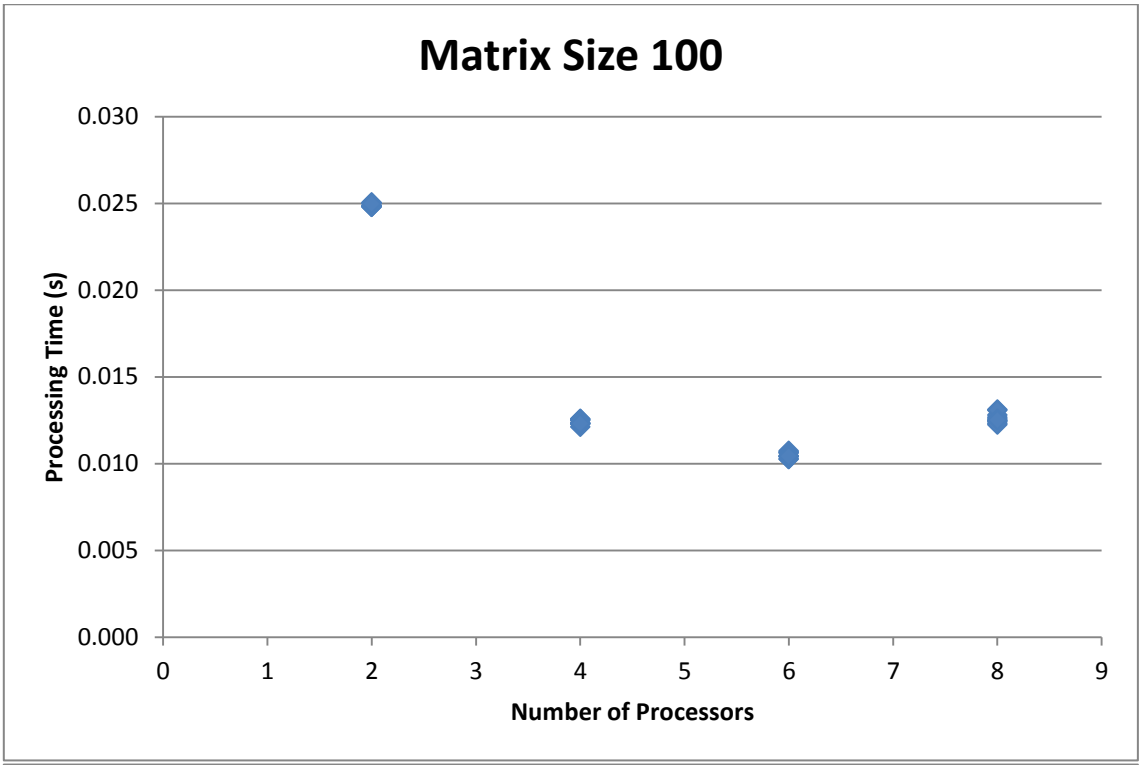


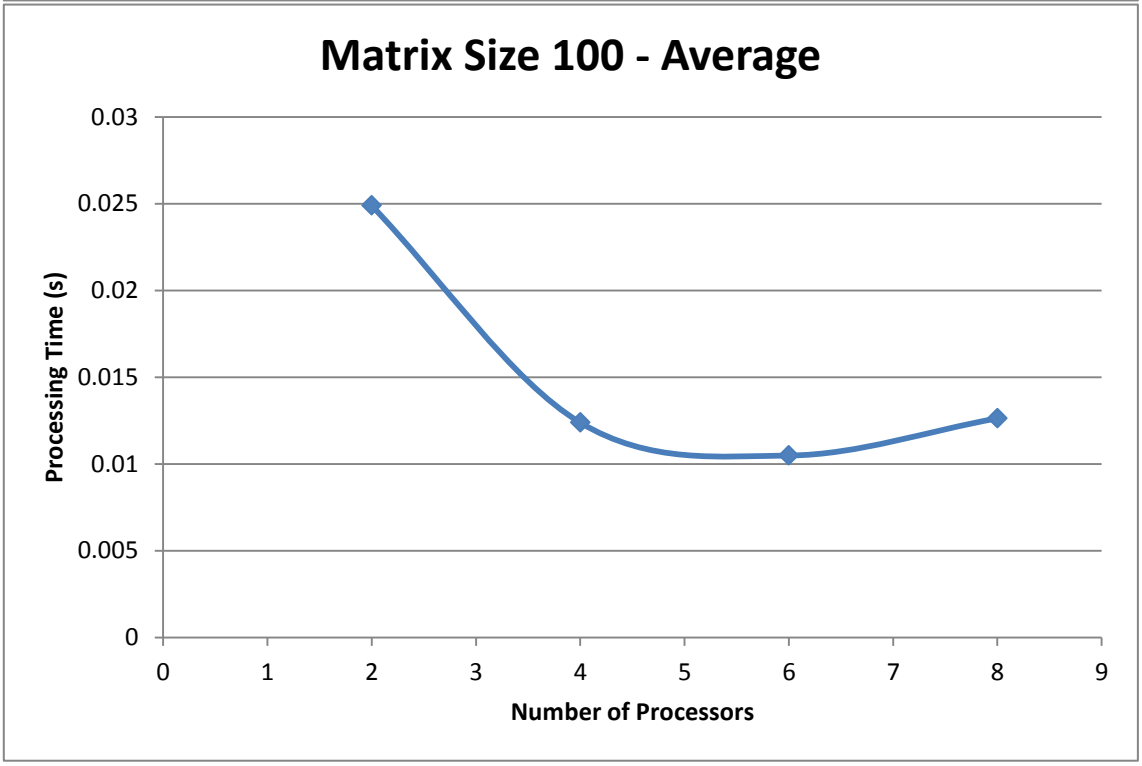
Home Computer Results:



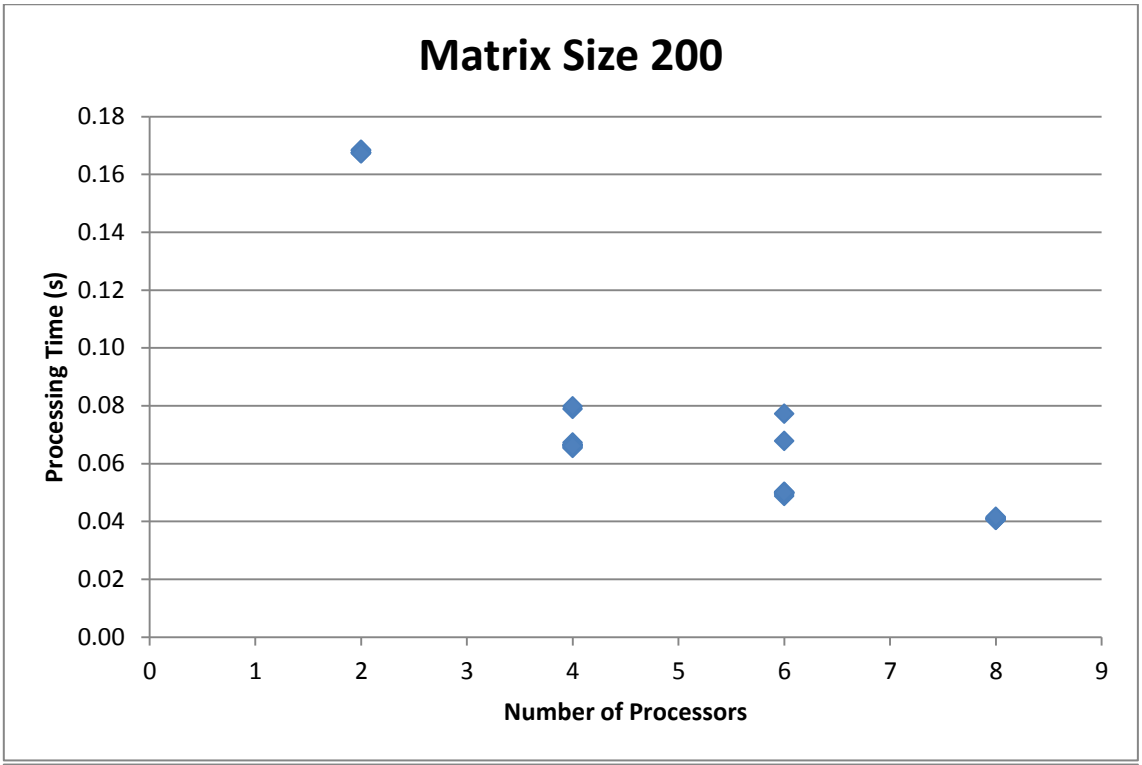


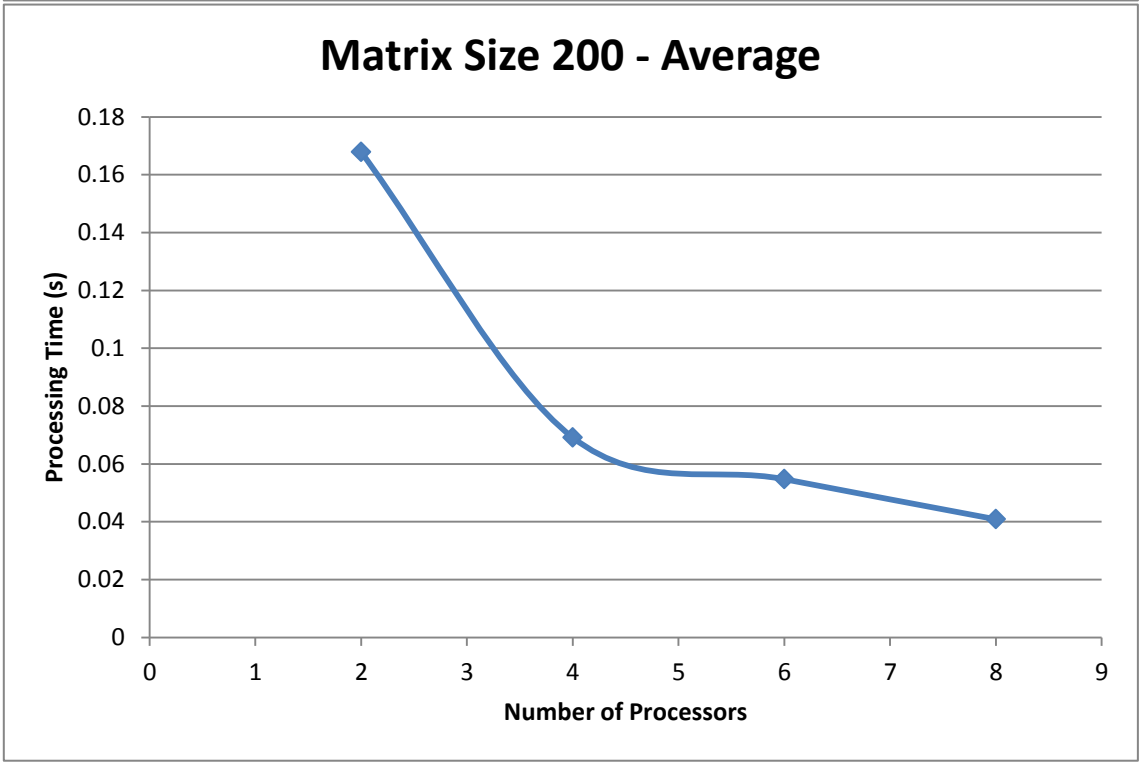
Manjra Cluster Results:



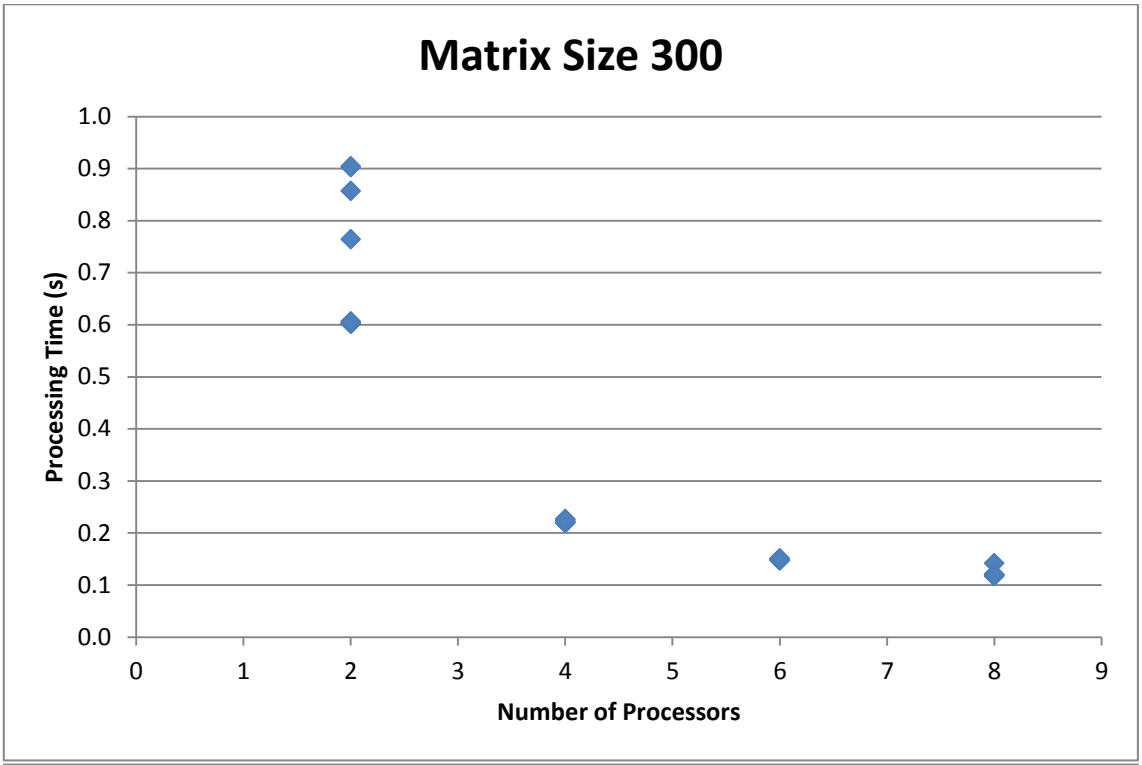


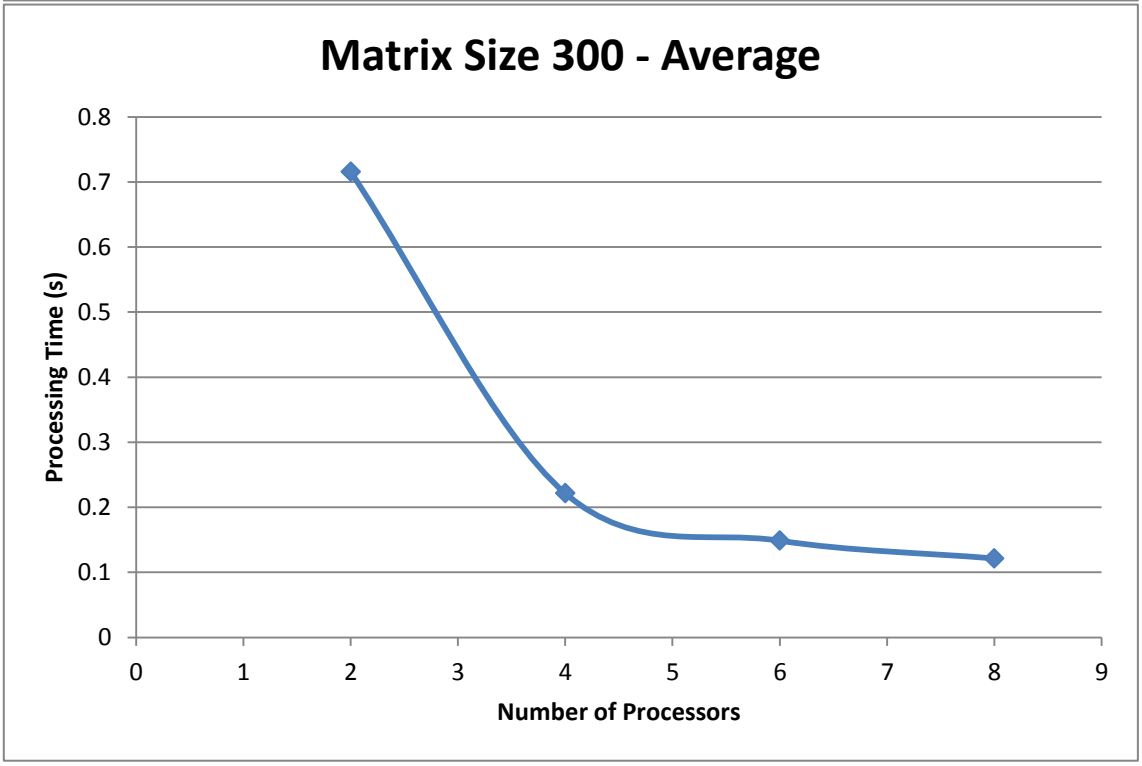
Manjra Cluster Results:



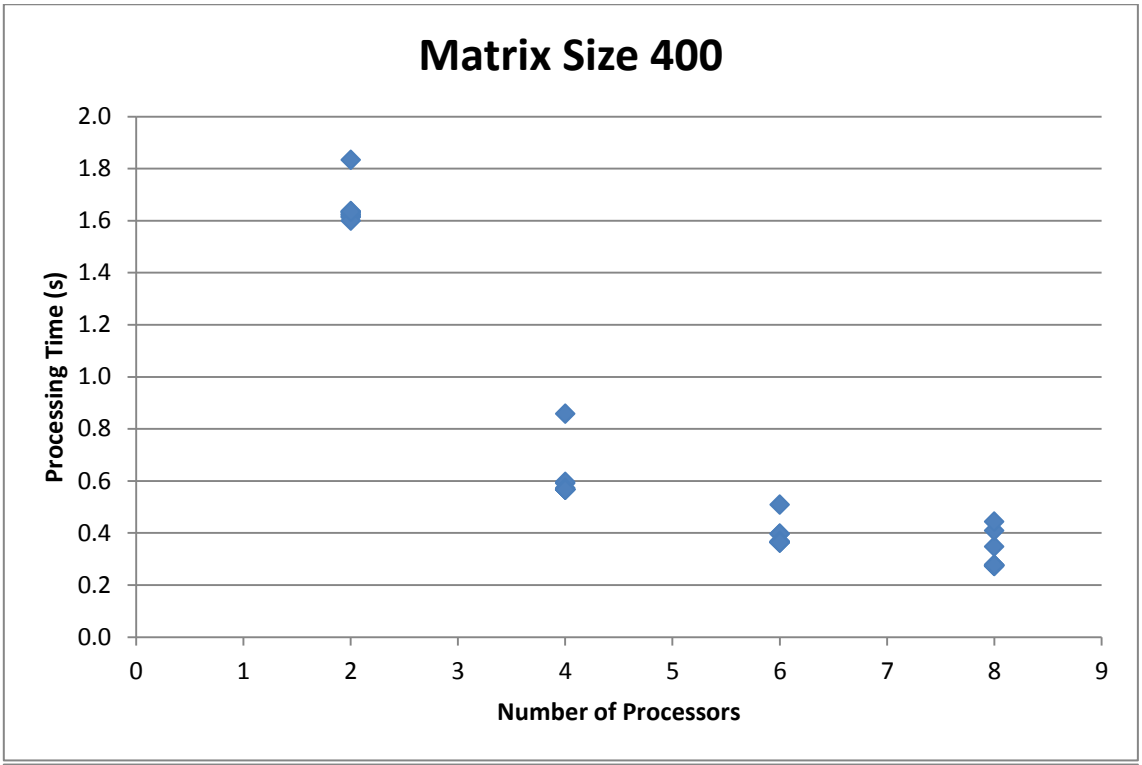


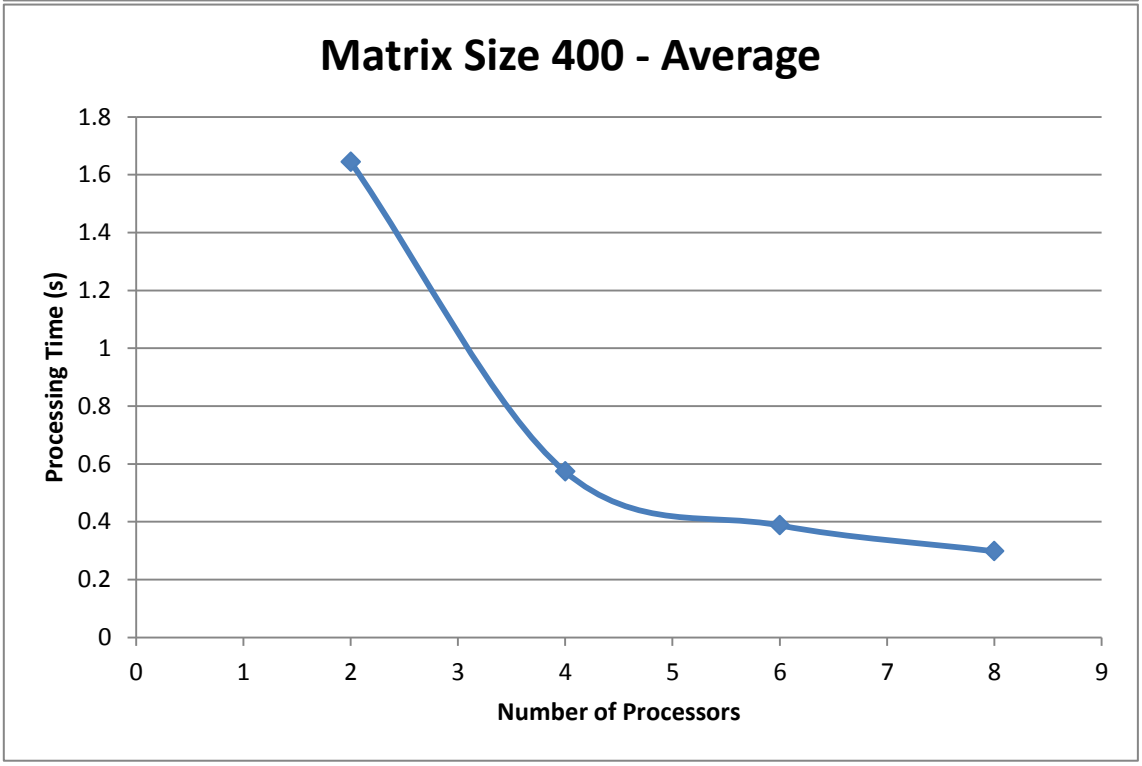
Manjra Cluster Results:





Manjra Cluster Results:





Manjra Cluster Results: