Project Assignment 04 Report

CS415

4-26-2017

By: Jervyn Suguitan

Table of Contents:

Description of Assignment

Project 04's main purpose was to teach students about matrix multiplication and how to parallelize the code.. This report will only talk about running matrix multiplication sequentially and how to do it and discussion based on several variables. The timing function that was used to time calculation for this project was MPI_Wtime.

Part 1: Sequential

The first part of the project tasks students in figuring out how to code matrix multiplication sequentially. The first thing to do is to create three 2-D arrays which act as the 3 matrices needed for the multiplication. Two of the matrices will be filled with numbers and act as the matrices that will be multiplied while the third matrix is used to hold the result. The three matrices are put into a function in which the actual matrix multiplication occurs. All matrices are of the same size, as in the width and length of the matrices are equal. The sizes of the matrices are all multiples of 120 due to the fact that for the parallel code, also uses a perfect square amount of processors (4, 9, 16, 25). The matrices are multiples of 120 as 120 can be divided by those perfect square amounts. The runtime of matrix multiplication is O(n^3).

Figure 1, Figure 2 and Figure 3 showcase the runtime of matrix multiplication with multiple different matrix sizes. The only difference between Fig. 1, Fig. 2 and Fig. 3 is that Fig. 1 includes the trials with all matrix sizes while Fig. 2 and Fig. 3 showcase a close look to each set of tests. These matrix sizes were chosen to show times over a large variety of sizes. These graphs show that as more numbers that are needed to be sorted, the longer it takes to sort. To specify all of the trials in Fig. 1 and Fig.2 all use a number range of 0 to 10000. Thus the numbers that are sorted will only be between that range.

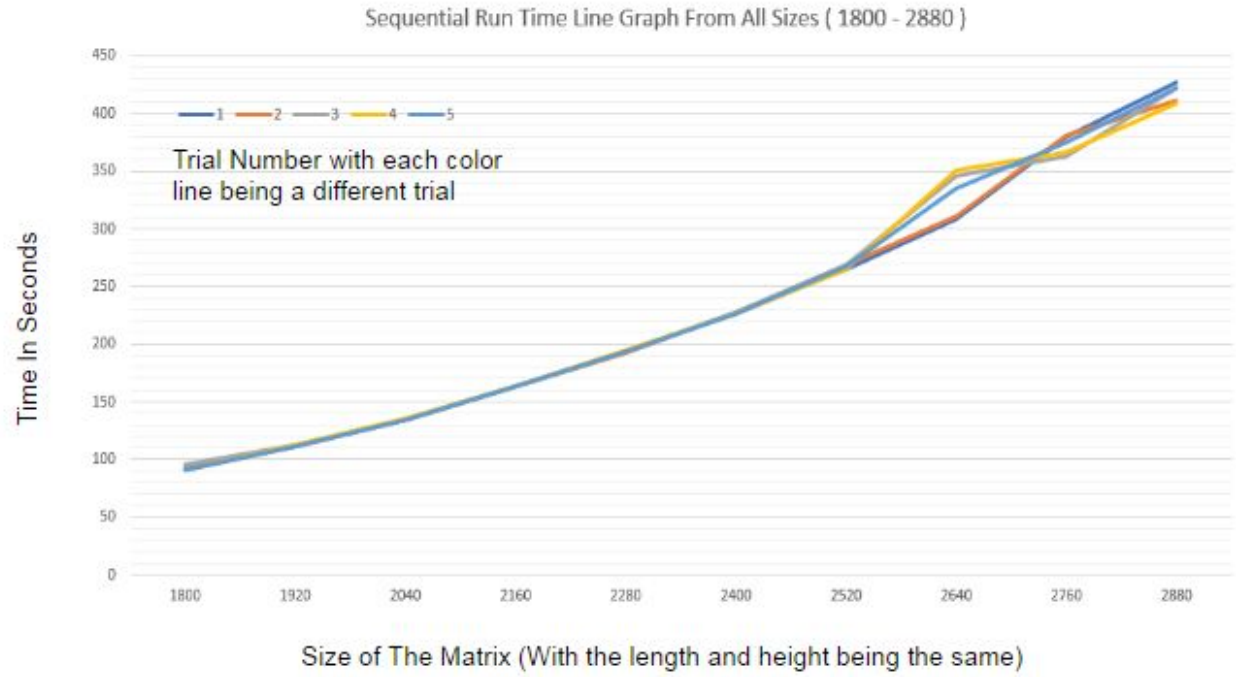


Fig. 1: Graphs that showcase the sequential times for different matrix sizes when doing matrix multiplication. The Y axis shows the time in seconds and the x-axis shows the matrix size. This graph includes times for 2880 by 2880 sized matrix. The times do not vary large amounts since everything is sequential thus there can not be a large speed up. Speed up or slow down may occur due to the processor speed which could be different for every trial. Some of the trials are hard to see as they are behind other lines.

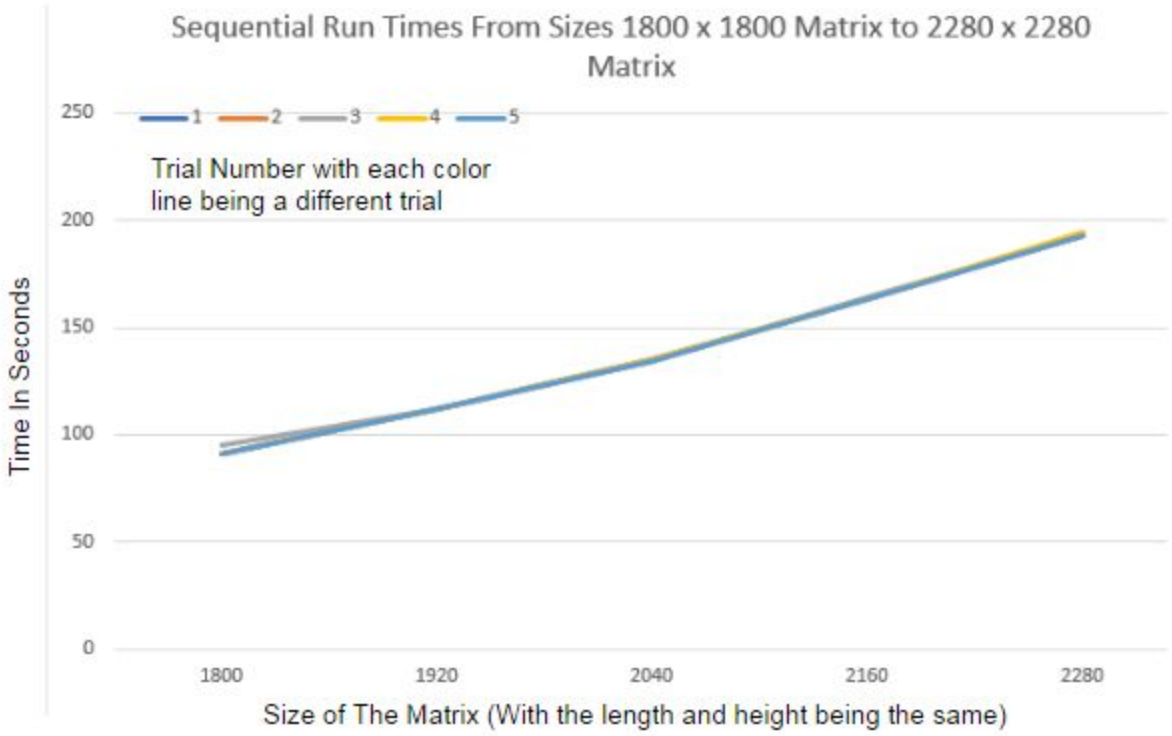


Fig. 2: Graphs that showcase the sequential times for different matrix sizes when doing matrix multiplication. The Y axis shows the time in seconds and the x-axis shows the matrix size. This graph includes times for 1800 to 2280 sized matrix. The times do not vary large amounts since everything is sequential thus there can not be a large speed up. Speed up or slow down may occur due to the processor speed which could be different for every trial. Some of the trials are hard to see as they are behind other lines.

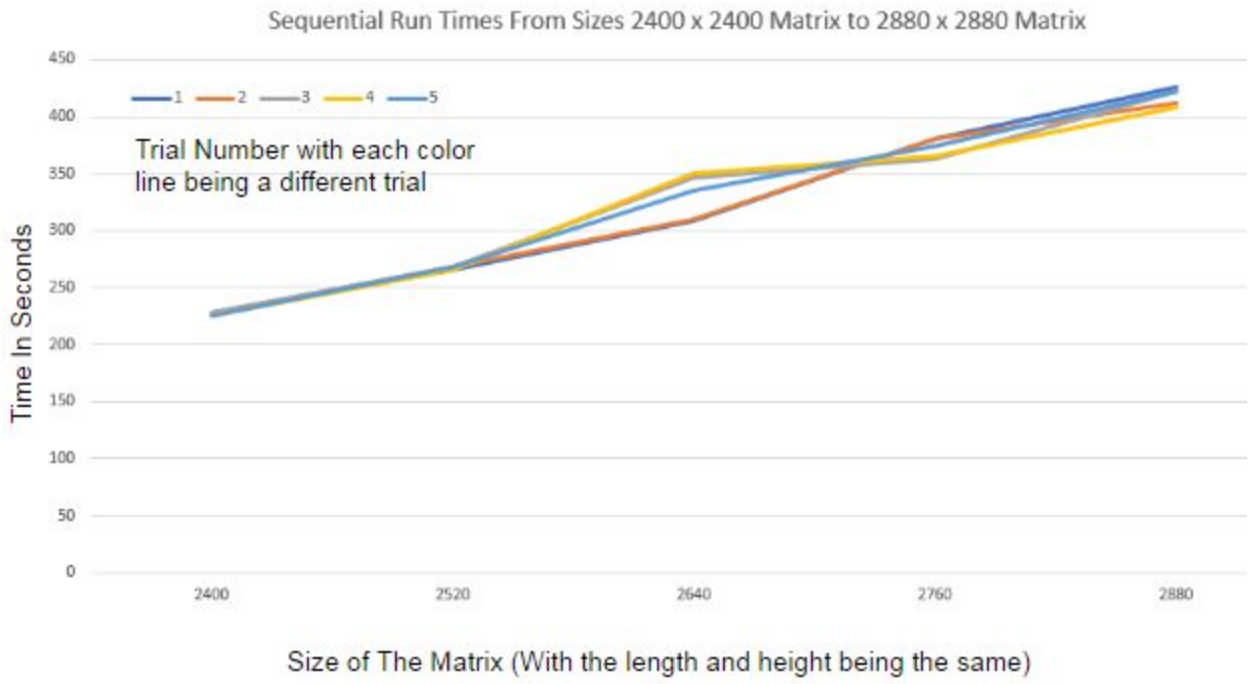


Fig. 3: Graphs that showcase the sequential times for different matrix sizes when doing matrix multiplication. The Y axis shows the time in seconds and the x-axis shows the matrix size. This graph includes times for 2400 to 2880 sized matrix. The times do not vary large amounts since everything is sequential thus there can not be a large speed up. Speed up or slow down may occur due to the processor speed which could be different for every trial. Some of the trials are hard to see as they are behind other lines.

Changes From Critics

One of the main comments all the commenters said about my code is that there is some repeated code for two of the functions, initRow and initCol. While they are exactly the same code with one line difference I did this initially to showcase that there are two moves one being a matrix move for rows and matrix move for columns. To fix this problem I did two things. First I made one move function (called move_MatrixRowOrCol) that can do both row and column matrix movement. To choose between where to move Rows and Columns there is an bool argument that is inputted in the function. Previously inside the initRow and initCol functions there was repeated code for the actual moving of the matricies. I changed to make the moves more modular by making a moveHelper function. This just does the same code as initRow

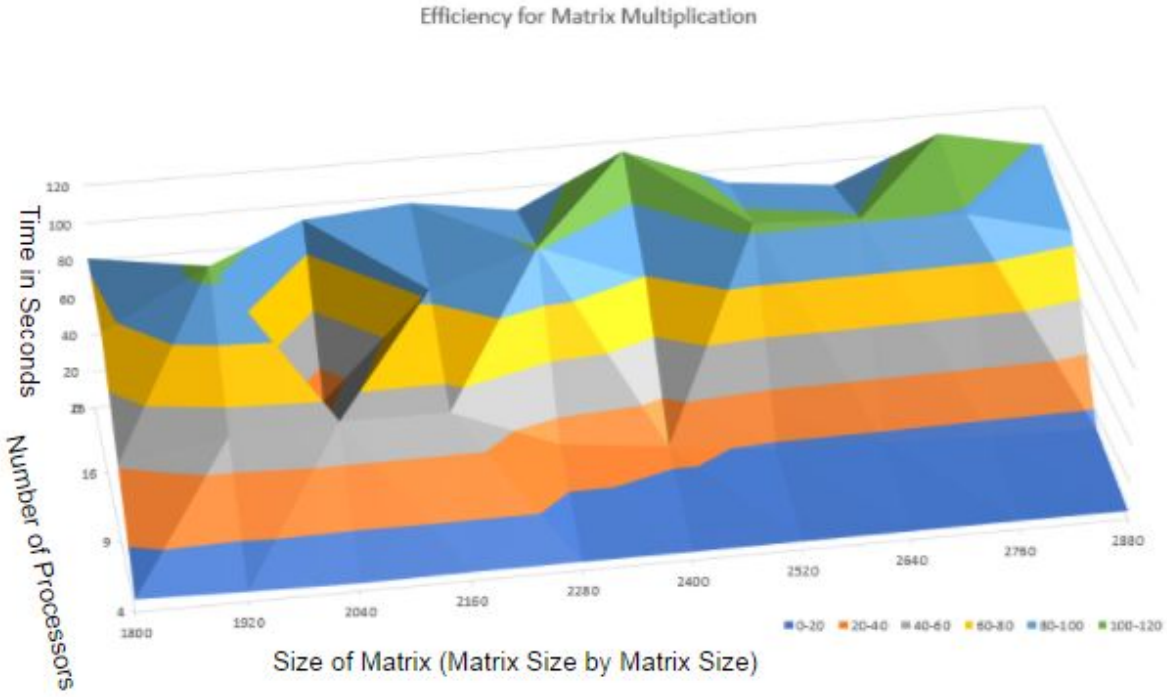
did but separates it from the main move_MatrixRowOrCol function so it is more readable.

Another problem one of the commenters talked about is the moving data from a small sub matrix to a bigger final matrix or inputting data to a smaller sub matrix from a big matrix. I did not realize when writing my code that they were very similar so I made an inputArray function that can work for both types of cases. Also for readability I created a create2DMatrix function as some parts of the code need a buffer 2D matrix to work and that code to do this is reoccuring throughout the code.

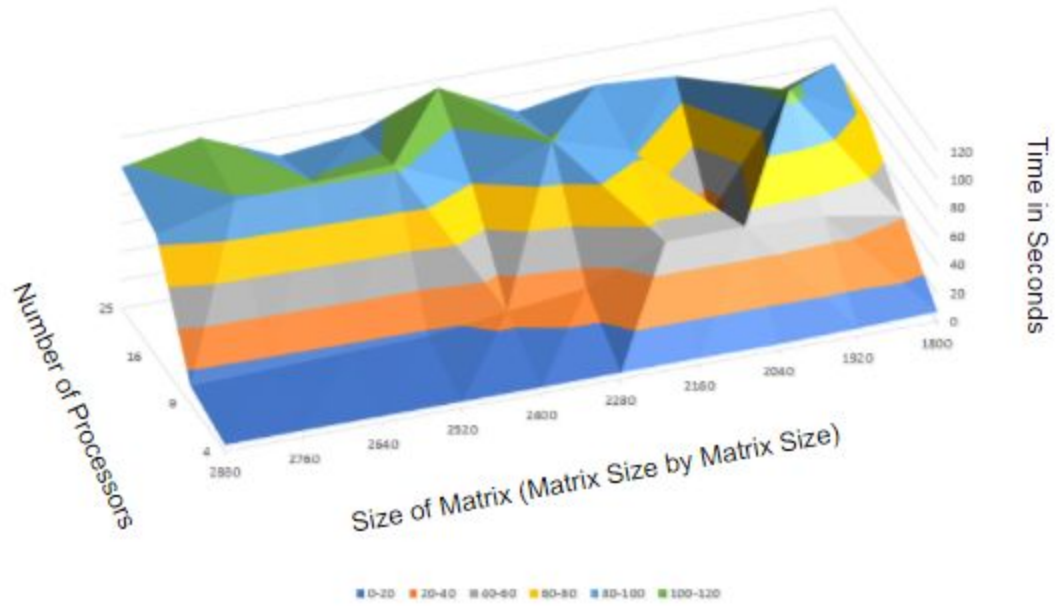
	Average Run Time									
	1800	1920	2040	2160	2280	2400	2520	2640	2760	2880
1	92.06792	111.9422	134.8292	163.39	193.3766	226.590833	266.415333	331.107167	370.929667	419.834167
4	13.654	21.4605167	32.65892	35.81605	42.90833	52.504975	59.54175	74.0511333	80.4113167	92.7416333
9	2.24691	2.20557667	2.524142	3.087543	6.319663	9.3366775	15.813875	20.9758	26.413925	33.6204
16	1.21637	1.05762183	4.71214	1.956687	1.915218	2.8684825	2.563775	3.25877	3.7422375	4.9226425
25	1.1329175	1.74787833	1.462007	1.69395	2.201753	1.9784525	2.8635075	3.8056675	3.374415	4.2398425

Speed Up										
	1800	1920	2040	2160	2280	2400	2520	2640	2760	2880
4	6.74292661	5.21619315	4.128404	4.561921	4.506738	4.31560692	4.47442901	4.47133152	4.61290378	4.52692228
9	40.9753484	50.7541641	53.41586	52.91911	30.5992	24.2688937	16.8469356	15.7851985	14.042959	12.4874828
16	75.6907191	105.843314	28.61316	83.50339	100.9685	78.9932772	103.915255	101.604951	99.1197557	85.286341
25	81.2662175	64.0446179	92.22201	96.45503	87.82849	114.529327	93.0381126	87.0037035	109.92414	99.0211704

Efficiency										
	1800	1920	2040	2160	2280	2400	2520	2640	2760	2880
4	1.68573165	1.30404829	1.032101	1.14048	1.126685	1.07890173	1.11860725	1.11783288	1.15322595	1.13173057
9	4.55281649	5.63935156	5.935096	5.879901	3.399911	2.69654374	1.87188173	1.75391094	1.56032878	1.38749809
16	4.73066994	6.61520714	1.788322	5.218962	6.31053	4.93707983	6.49470345	6.35030945	6.19498473	5.33039631
25	3.2506487	2.56178472	3.688881	3.858201	3.51314	4.58117308	3.72152451	3.48014814	4.3969656	3.96084682


Efficiency for Matrix Multiplication
Time in Seconds
120
100
80
60
40
20
Number of Processors
25
16
9
4
1800
1920
2040
2160
2280
2400
2520
2640
2760
2880
Size of Matrix (Matrix Size by Matrix Size)
0-20
20-40
40-60
60-80
80-100
100-120

Speed Up



Average Run Time in Seconds