PA4 – Matrix Multiplication Documentation

Brian Conway

2 May 2017

CS415

Matrix Multiplication - Sequential

This sequential implementation of matrix multiplication involves taking the dot product of each row in matrix A and with each column of matrix B in order to calculate the values of each cell in matrix C.

Methodology

The file sequential.cpp is responsible for generating random values for square matrices A and B based on the dimensions specified, matrix multiplying them, and outputting statistics about the timing as well as optionally writing the three matrices to a file.

The program has one mandatory command line parameter for the dimensions of the square matrix, and an optional parameter for if you want the matrices output to a file. If the second command line parameter is “y”, then the results are output to the file “output.txt”.

The program first calls generateNumbers(), a function which uses c++11 random devices to generate uniformly distributed values from 0 to 999,999 for all of the cells of matrices A and B. The generator is seeded appropriately so that this sequential version generates the same numbers as the eventual parallel one.

Afterwards, the program calls the matrixMult() function, which loops through each row of A and column of B, then loops through each element in that row/column combination to sum up the multiplication of all corresponding elements and store the result in the corresponding cell in C. For example, to calculate the cell in row 3, column 2 of C, it takes the dot product of row 3 of matrix A and column 2 of matrix B.

It times this using the Timer class, and it repeats this process the amount of times specified by the NUM_MEASUREMENTS constant. Afterwards, it calls the calcStatistics() function to get an average of the time taken to multiply the matrices and outputs it to the console.

Timing is done using the custom Timer class, which has start, stop, and resume functions similar to a stopwatch. Timer makes use of timevals and the gettimeofday() function in order to get an accurate reading of seconds and milliseconds at certain instants. When the elapsed time is to be given, the Timer takes the difference in seconds and milliseconds between when the timer was started and when the timer was stopped. It then combines the seconds and milliseconds reading into a double precision floating point variable.

Results – Sequential

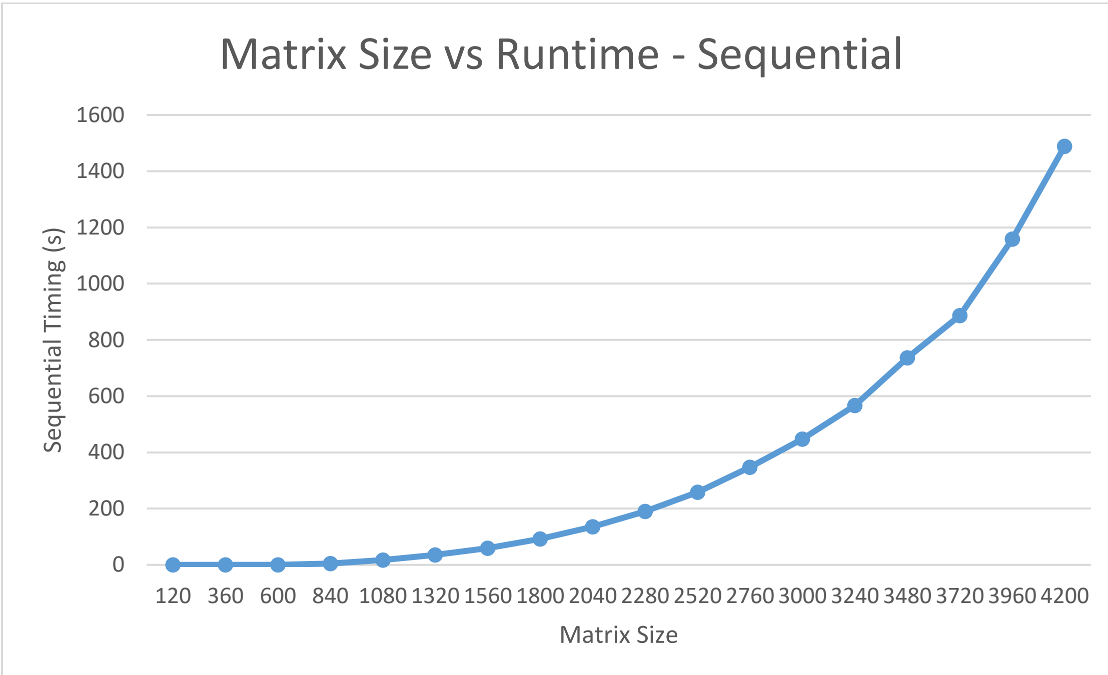

Fig. 1 Graph of sequential runtimes with different sizes of the matrices.

120	**360**	**600**	**840**	**1080**	**1320**	**1560**	**1800**	**2040**
0.006546	0.145585	0.716863	4.41264	17.2486	35.3099	58.9637	92.0221	135.221

Table 1: First half of timings for sequential matrix multiplication calculations, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

120	**360**	**600**	**840**	**1080**	**1320**	**1560**	**1800**	**2040**
190.317	258.853	347.023	447.536	567.295	736.911	886.414	1158.14	1489.15

Table 2: Second half of timings for sequential matrix multiplication calculations, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

Results – Parallel

Parallel – 4 process mesh

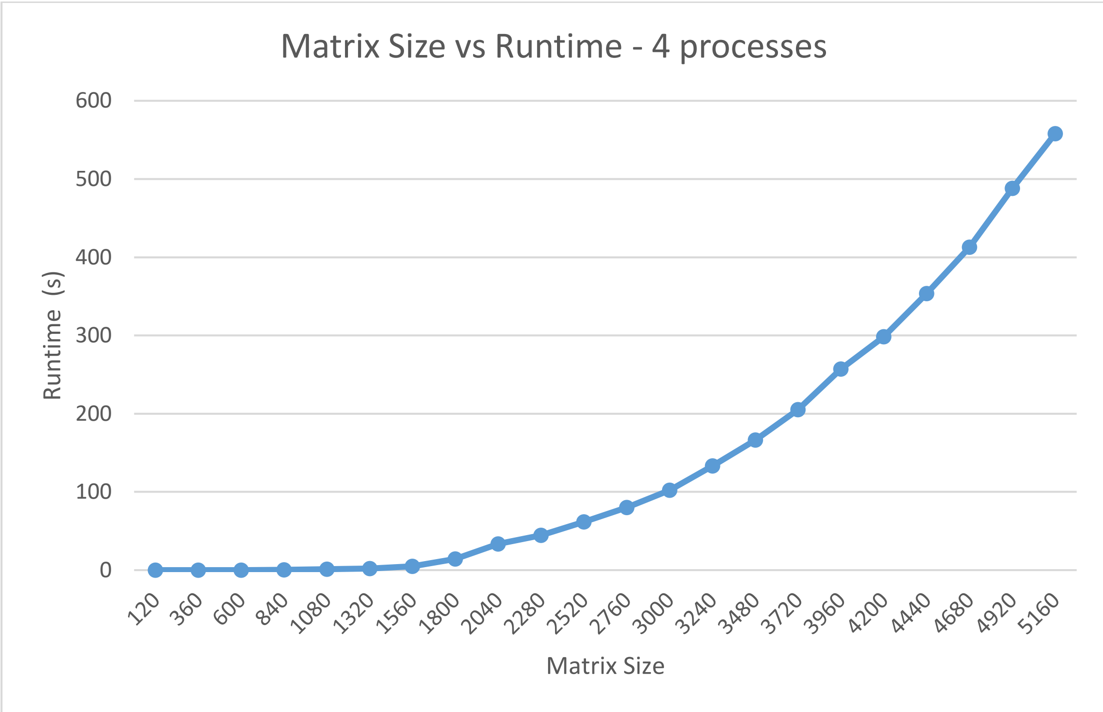

Fig. 2 Graph of runtimes with different sizes of the matrices for a 4 process mesh.

120	360	600	840	1080	1320	1560	1800	2040
0.003484	0.021125	0.090451	0.293056	1.16535	1.99225	5.03024	14.4529	33.3656

Table 3: First part of timings matrix multiplication calculations with 4 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

2280	2520	2760	3000	3240	3480	3720	3960	4200
7.30272	16.048	27.1116	41.8183	56.834	72.3295	90.2638	112.372	131.245

Table 4: Second part of timings for matrix multiplication calculations with 4 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

4440	4680	4920	5160
353.809	413.03	488.305	558.123

Table 5: Last part of timings for matrix multiplication calculations with 4 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

Parallel – 9 process mesh

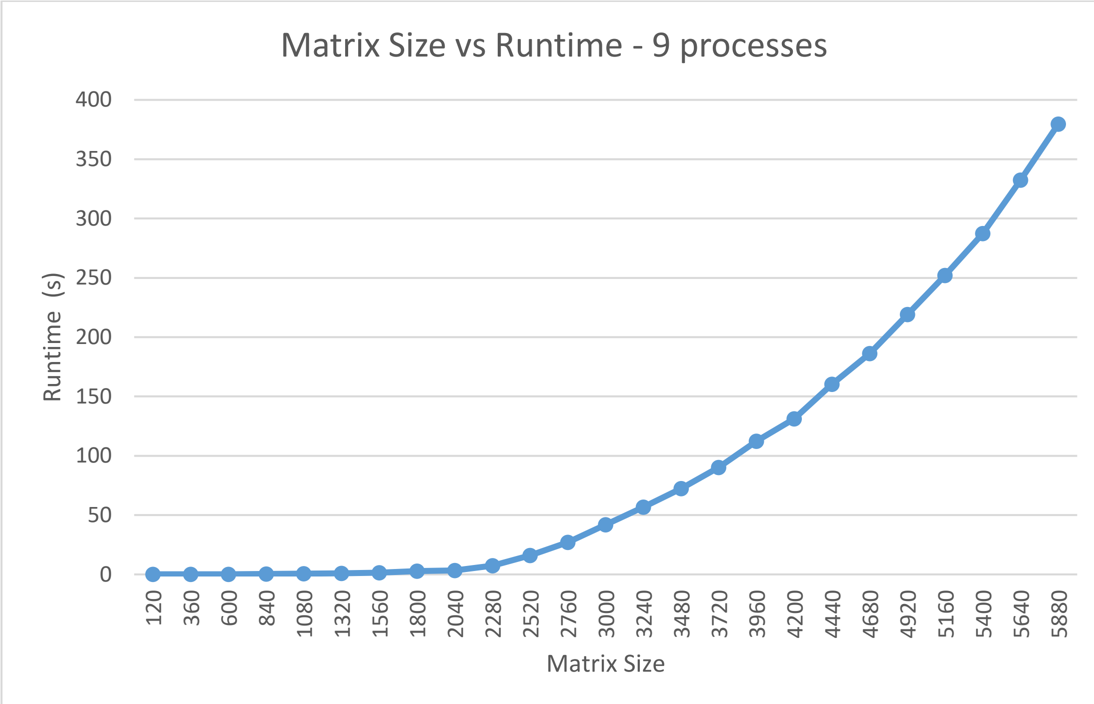

Fig. 3 Graph of runtimes with different sizes of the matrices for a 9 process mesh.

120	360	600	840	1080	1320	1560	1800	2040
0.112949	0.125255	0.135314	0.313103	0.690703	0.93117	1.4113	2.76752	3.34771

Table 6: First third of timings matrix multiplication calculations with 9 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

2280	2520	2760	3000	3240	3480	3720	3960	4200
7.30272	16.048	27.1116	41.8183	56.834	72.3295	90.2638	112.372	131.245

Table 7: Second third of timings for matrix multiplication calculations with 9 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

4440	4680	4920	5160	5400	5640	5880
160.431	186.143	219.277	251.983	287.508	332.462	379.619

Table 8: Last third of timings for matrix multiplication calculations with 9 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

Parallel – 9 process mesh

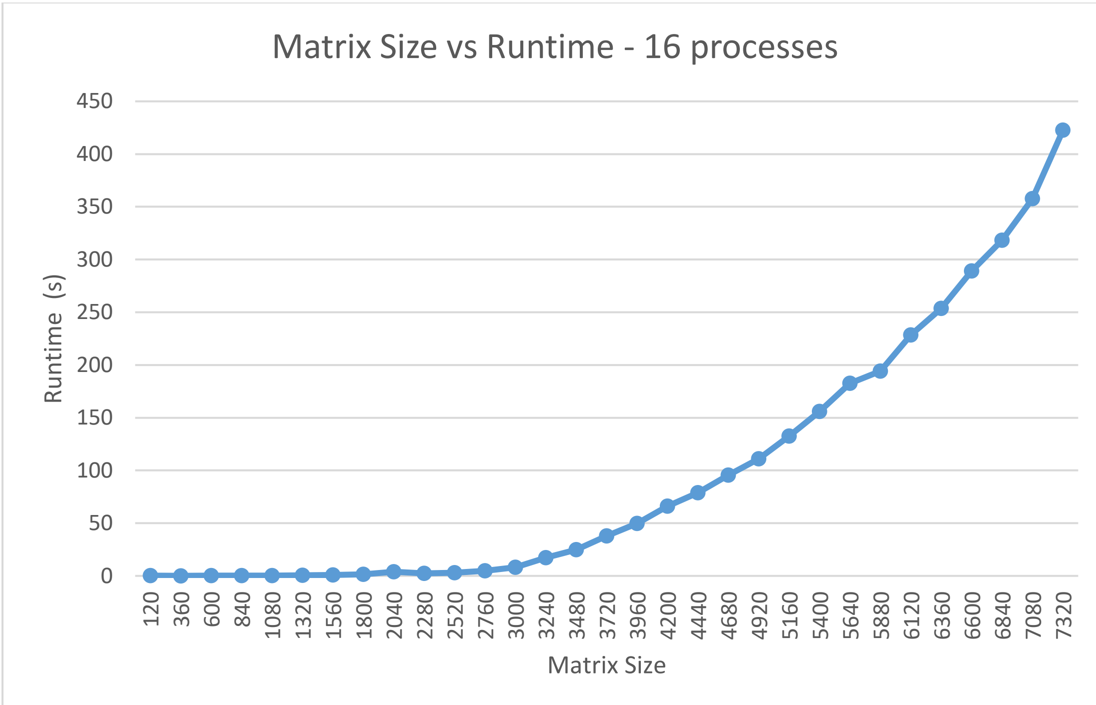

Fig. 4 Graph of runtimes with different sizes of the matrices for a 16 process mesh.

120	360	600	840	1080	1320	1560	1800	2040
0.172028	0.048591	0.117823	0.153232	0.247421	0.491062	0.754605	1.44936	3.93138

Table 9: First part of timings matrix multiplication calculations with 16 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

2280	2520	2760	3000	3240	3480	3720	3960	4200
2.28094	3.11749	4.69951	8.08982	17.1978	24.7959	37.9716	49.7614	66.0482

Table 10: Second part of timings for matrix multiplication calculations with 16 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

4440	4680	4920	5160	5400	5640	5880	6120	6360
78.8248	95.48	111.111	132.495	156.039	182.562	194.174	228.423	253.517

Table 11: Last part of timings for matrix multiplication calculations with 16 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

6600	6840	7080	7320
289.097	318.358	357.77	422.593

Table 12: Last part of timings for matrix multiplication calculations with 16 processes, with the top row being the sizes of the matrices and the bottom row indicating the runtimes in seconds.

Analysis

Sequential

As for the sequential algorithm, the main goal was to find out how the runtimes would look with different square matrix sizes, up to five minutes. With the $O(n^3)$ algorithm being used for matrix multiplication, it got to five minute runtimes at around the 2600x2600 matrix sizes.