Ryan Devaney
5/3/2017
CS 415
PA04 - Matrix Multiplication

Introduction

This is the report for project 4, matrix multiplication. This goal of this project was to implement matrix multiplication in both sequential and parallel and compare the time it takes to complete the multiplication. The sizes of the matrices increased from 120 until it reached 5 minutes to complete sequentially. The sizes were incremented by 120 each time in order to give a large range of sizes. The size got up to 2640 rows/columns before reaching that 5 minute mark, and the parallel processes were tested up until that point as well.

Sequential

The sequential portion of the code was pretty straight forward. A random number generator was used to fill in the matrices, and 3 for loops were used in order to do the calculation. The timer for this started right before entering the loops and ends immediately after leaving. Thus ensuring that we are not timing I/O operations, but just the pure calculation. Below you will see a table showing the times for the various tests of sequential matrix multiplication.

Amount of Rows and Columns	Average time Taken(seconds)
120	0.006490851
240	0.02870466
360	0.1757363
480	0.3131466
600	1.421064
720	3.052699
840	6.524932
960	10.95011
1080	16.82777
1200	24.304
1320	36.26744
1440	44.92931
1560	57.818
1680	74.239
1800	92.01015
1920	113.837
2040	135.938
2160	163.1867
2280	194.7875
2400	232.1
2520	267.1188
2640	333.6505

Chart 1: Timings for the various tests of sequential matrix multiplication.

As shown from this chart, the times are incredibly fast at the beginning and grow extremely fast. It will be very apparent from looking at the graph below just how fast this operation ramps up.

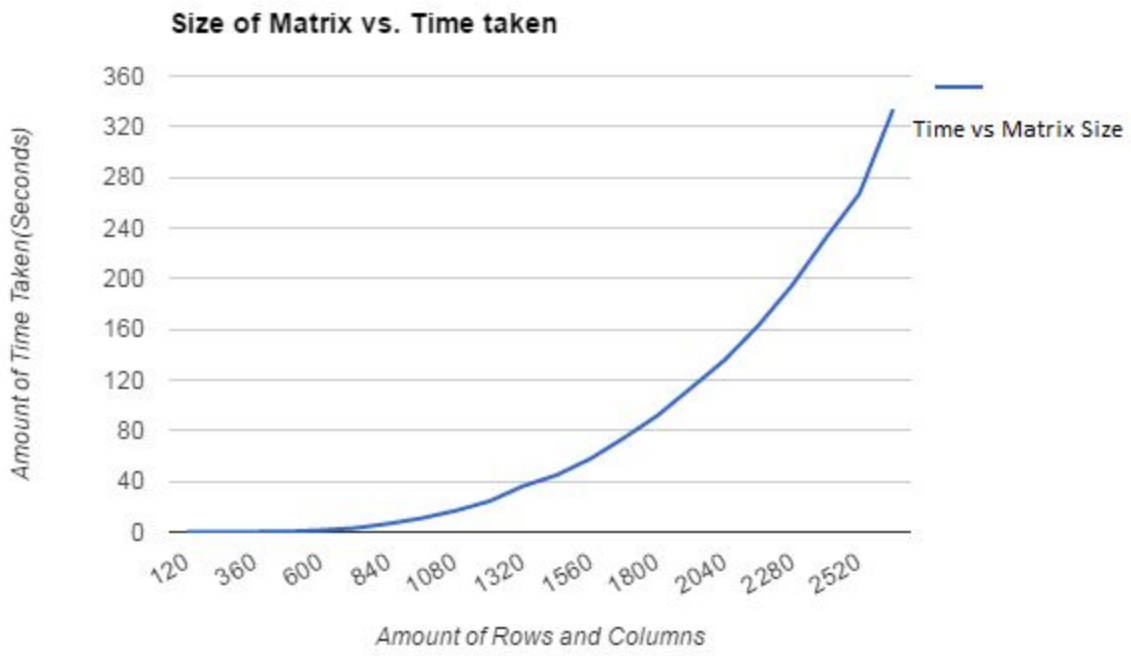

Fig. 1: Graph showing Amount of time taken against the size of the matrix in a NxN format.

As you can see from the graph there is a large amount of growth as the size of the matrices increase. The graph shows an exponential trend which makes sense as matrix multiplication has a growth rate of O(n^3). This graph and data are very important as they will be used to compare to the parallel timings.

Parallel

The parallel matrix multiplication was much more interesting than the sequential version of it. In order to properly parallelize the code, cannon's algorithm had to be used. Cannon's algorithm is based on shifting rows and columns in order to segment the matrix into a smaller portion. These smaller portions could be sent out to the various processor and then they could be calculated in order to get the proper matrix back to the main processor. Another important thing to note is that the number of processors used had to be the square of a number. For this reason the amount of processors used were 4, 9, 16, and 25.

Parallel Improvement

Most of the comments I received for my code review was to fix small formating issues. In addition, I had to set a flag for printing out the matrices instead of just commenting the code out. Another thing that was changed after the peer review was being able to take input files for the matrices. The review helped me make my code more manageable and easier for people to

read. In addition, it got me to properly output the matrices and because of that I found a bug in my row and column shifting. Overall, the code review was very beneficial as I got some tips on my code, and the act of reviewing other people's code showed me why it is so important to make it readable and have comments to help people understand it.

Parallel Timings

Run Time	4 Processors	9 Processors	16 Processors	25 Processors
120	0.004761695	0.004971	0.005719	0.27942
240	0.009226085	0.009943985	0.0101987	0.0084407
360	0.0261494	0.01915275	0.01213095	0.0193489
480	0.0488892	0.0427158	0.02464215	0.027308
600	0.107684	0.06467065	0.035969	0.041307
720	0.297458	0.1068438	0.0569532	0.05983
840	0.32489	0.148534	0.0918109	0.084687
960	0.453428	0.25845	0.133853	0.104983
1080	0.921445	0.499328	0.177032	0.163976
1200	1.25801	0.705017	0.229788	0.212825
1320	1.7354	0.625194	0.330892	0.294921
1440	2.4897578	0.804685	0.4186	0.335108
1560	4.5285	1.034	0.554306	0.516628
1680	8.93156	1.37049	0.65128	0.59138
1800	13.6461	2.0694448	1.18796	0.89854
1920	21.5219	2.1096221	1.34537	1.0685
2040	31.6298	2.7228	3.88122	1.2471
2160	36.6729	3.0917982	2.31981	1.69038
2280	44.6549	6.9152	3.4349	2.10037
2400	52.0332	9.67009	5.1983	2.04138
2520	58.94318	17.049	8.3018	2.65366
2640	73.205213	21.2285	11.19807	3.31647

Chart 2: Parallel run times for various sizes and processors.
These are the run times for all the tests run and on the different number of processors.

The amount of processors heavily affects the amount of time taken to compute the matrix. While at the low end of the spectrum we have a huge variance in times, at larger numbers we see a drastic reduction in time taken.

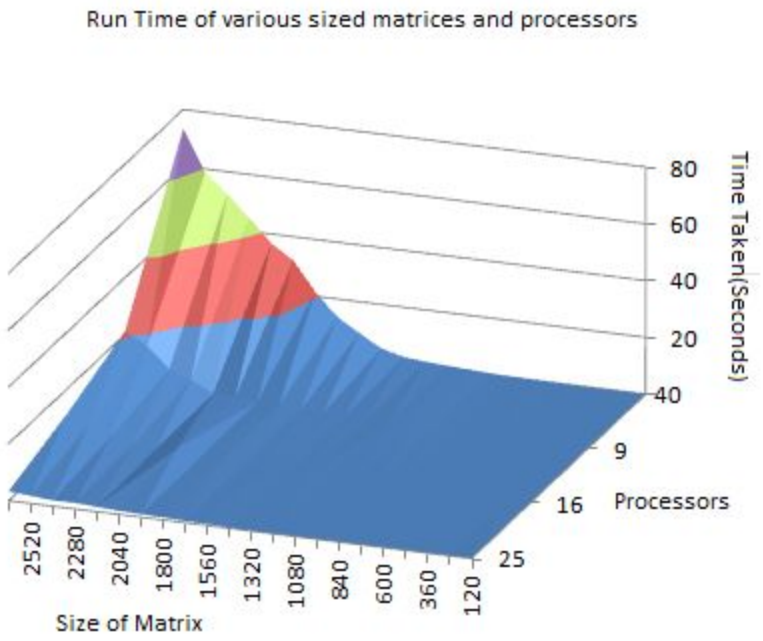

Fig 2: Surface graph showing the runtime of various sizes and processors

This 3-D graph shows the run times of the different test ran. At 4 processors it still takes a long time to complete, but as it gets closer to 25 processors, the amount of time goes down by a huge margin.

Parallel Speedup

Speedup	4 Processors	9 Processors	16 Processors	25 Processors
120	1.363138756	1.305743512	1.134962581	0.02322972944
240	3.11125033	2.886635489	2.814541069	3.400744014
360	6.720471598	9.175512655	14.48660657	9.082495646
480	6.405230603	7.330931412	12.70776292	11.46721107
600	13.19661231	21.97386295	39.5080208	34.40249837
720	10.26262195	28.5716064	53.60013134	51.0228815
840	20.08351134	43.92887824	71.06925213	77.04762242
960	24.1496114	42.36838847	81.80698229	104.3036492
1080	18.26237052	33.70083392	95.05496181	102.6233717
1200	19.31940128	34.4729276	105.7670549	114.1971103
1320	20.89860551	58.00989773	109.6050675	122.9734064
1440	18.04565488	55.8346558	107.332322	134.0741194
1560	12.76758308	55.91682785	104.3070073	111.914182
1680	8.311985812	54.16967654	113.9893748	125.5351889
1800	6.742596786	44.4612729	77.45222903	102.3996149
1920	5.289356423	53.9608492	84.61389804	106.5390735
2040	4.297782471	49.92581166	35.02455414	109.0032876
2160	4.449789899	52.78051459	70.34485583	96.53847064
2280	4.36206329	28.16802117	56.70834668	92.73961254
2400	4.460613608	24.00184486	44.64921224	113.6975967
2520	4.531801644	15.66771072	32.17601002	100.6605217
2640	4.557742356	15.71710201	29.79535759	100.6041062

Chart 3: The speedup of parallelizing matrix multiplication.

The speedup was calculated using the time it took to run sequentially and dividing it by the time it took to run parallel. Surprisingly the only slowdown was for 25 processors. This is surprising because at low sizes the time it takes to communicate between the processors is usually more than the time it actually takes to do the computations. It would make sense that an algorithm that has an n^3 growth rate would still take a lot of computations even at a small number. The second thing that is interesting here is the superlinear speedup. It is very prevalent after a 600x600 sized matrix. Another thing to note is that at 4 processors, the amount of speedup starts to slow down and almost gets below superlinear, and as the number of processors go up, there is not a much of a decline in the amount of speedup. This behavior can be explained by the cache in the processors. Because the matrix is being split into smaller sizes, the processors are able to hold them in cache and they don't need to swap data in and out of main memory.

This would explain why a larger processor count would be able to maintain the superlinear speedup as they are still able to fit their matrices in cache. Another thing to support this argument is when we look at much larger matrix sizes. For example, a 4080x4080 matrix on 4 processors took 271.799 seconds to compute. The time taken for sequential to run the same size can be estimated from getting an equation from the trendline based on the graph of the sequential times. The equation generated from the trendline is (1E-08)(x^3) + (8E-06)(x^2) - 0.0123*x + 1.802, where x is the size of the matrix. Plugging 4080 into the equation gives us a time of 763.96232. So the speedup in this case would be 2.811, which is no longer superlinear.

~5 minutes times	sequential	4	9	16	25	speedup
4080	763.96232	271.799				2.810762071
5280	1631.86472		266.152			6.131326159
6480	2978.99912			283.068		10.5239699
7440	4471.42664				262.16	17.0560979

Chart 4: Showing estimated times and actual times of 4,9,16, and 25 processors on various matrix sizes. As well as the speedup for these times.

As we can see from the chart above, there is no longer any superlinear speedup. Therefore the superlinear speedup can be explained by the split up matrices still being able to fit into cache.

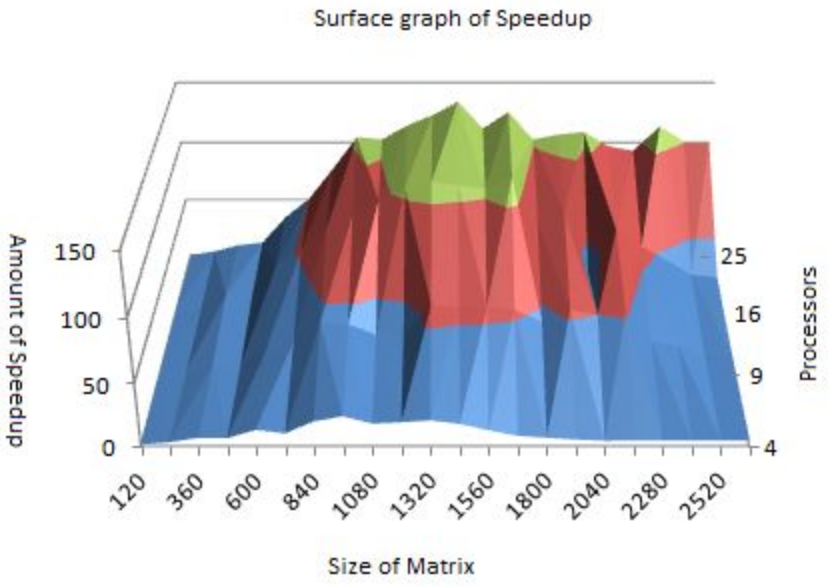

Fig. 3: Surface graph of Speedup of parallel vs. sequential matrix multiplication.

The graph shows that the largest amount of speedup is in the 840-1800 range for most of the processors. 25 processors maintains good speedup throughout all test ran.

Efficiency

Efficiency	4 Processors	9 Processors	16 Processors	25 Processors
120	0.3407846891	0.1450826125	0.0709351613	0.000929189177
240	0.7778125825	0.3207372765	0.1759088168	0.1360297606
360	1.680117899	1.019501406	0.9054129108	0.3632998258
480	1.601307651	0.8145479346	0.7942351824	0.4586884429
600	3.299153078	2.441540328	2.4692513	1.376099935
720	2.565655487	3.174622933	3.350008208	2.04091526
840	5.020877836	4.880986471	4.441828258	3.081904897
960	6.037402851	4.707598719	5.112936393	4.172145966
1080	4.565592629	3.744537102	5.940935113	4.104934869
1200	4.829850319	3.830325289	6.610440928	4.567884412
1320	5.224651377	6.445544192	6.85031672	4.918936258
1440	4.511413721	6.203850644	6.708270127	5.362964776
1560	3.191895771	6.212980873	6.519187958	4.476567279
1680	2.077996453	6.018852949	7.124335923	5.021407555
1800	1.685649196	4.940141433	4.840764314	4.095984597
1920	1.322339106	5.995649911	5.288368627	4.261562939
2040	1.074445618	5.547312407	2.189034633	4.360131505
2160	1.112447475	5.864501621	4.396553489	3.861538826
2280	1.090515822	3.12978013	3.544271667	3.709584502
2400	1.115153402	2.666871652	2.790575765	4.547903869
2520	1.132950411	1.740856746	2.011000626	4.026420868
2640	1.139435589	1.746344668	1.86220985	4.024164247

Chart 5: Efficiency for all processors and matrix sizes.

The efficiency was calculated by taking the speedup and dividing it by the number of processors. The efficiency chart shows similar trends to the speedup chart. The largest amount of efficiency is present after 600x600 sized matrices and begins to slow down afterwards based on the amount of processors. The same thing could be said for the cache contributing to the large efficiency for small sizes. The efficiency for the larger sizes is much more accurate to the actual efficiency of parallelizing matrix multiplication.

Processors	Efficiency
4	0.7026905176
9	0.6812584621
16	0.6577481188
25	0.682243916

Chart 6: Efficiency for estimated sequential times and larger parallel times.
As we can see, the efficiency is much closer to the values we would expect.

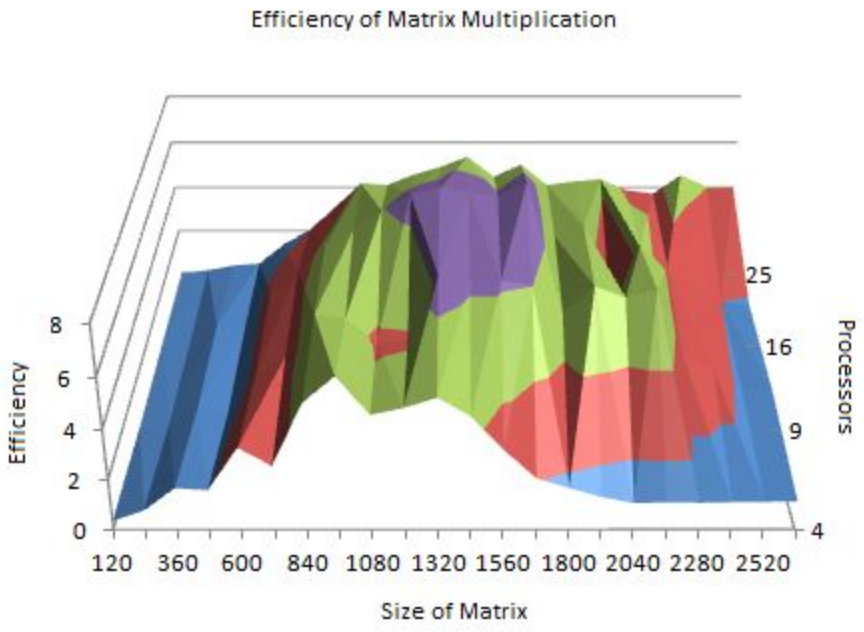

Fig. 4: Surface graph of the efficiency of parallelizing matrix multiplication.

This 3-D graph shows the highest efficiency in the same range as speedup. 25 processors was able to maintain its efficiency while the smaller number of processors saw a drop off. But as shown in the chart above as the size of the matrices get larger they all even out and come to similar numbers.

Conclusion

Parallelizing matrix multiplication shows dramatic changes in the speedup especially as larger sizes of matrices are used. It is unfortunate that we could not do unlimited testing as running very large matrices is required in order to get accurate data on parallelizing matrix multiplication. Due to the cache we saw superlinear speedup all throughout the smaller sizes, but when larger sequential sizes were estimated the numbers made much more sense. Doing this project and seeing how long it takes it calculate these relatively small matrices gave me a much deeper appreciation for how important having a HPC and good parallel code is for scientists as they need to calculate much larger matrices than what we have done.