Drake Seifert

CS 475 Project 5

What machine you ran this on

All tests were ran on FLIP.

Show the tables

Without vectorization:

Note: Obviously the performance for 1000 data set size didn't see an infinite performance. The calculations were just too quick for the omp_get_wtime() function to capture.

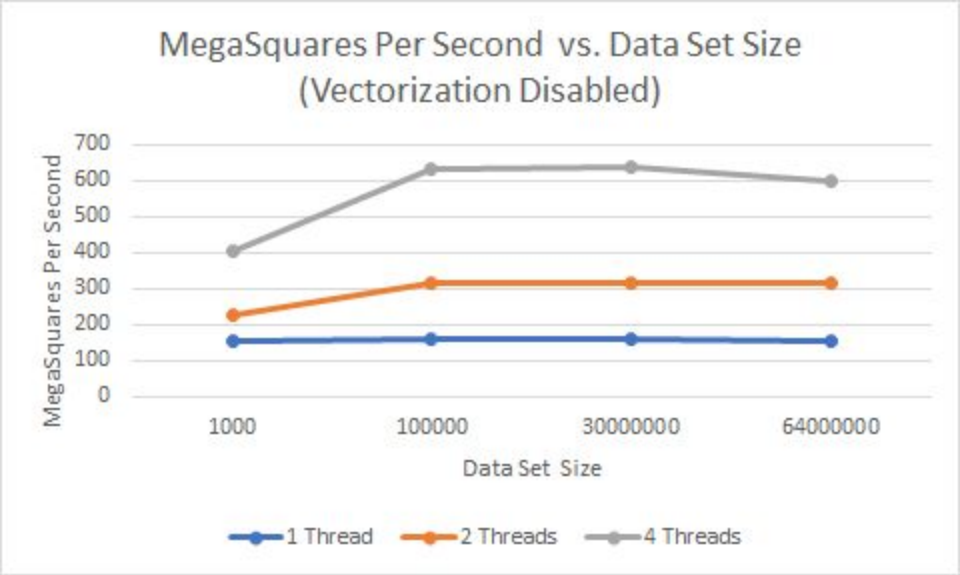
	1000	100000	30000000	64000000
1	155.11	159.68	158.29	156.49
2	229.13	314.38	315.35	314.22
4	402.96	630.47	635.98	596.29

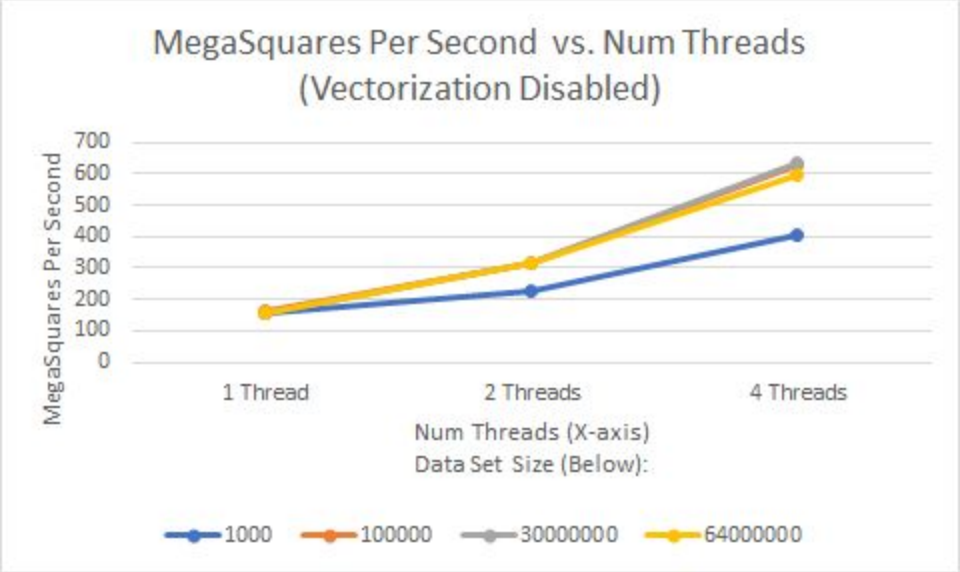
With vectorization:

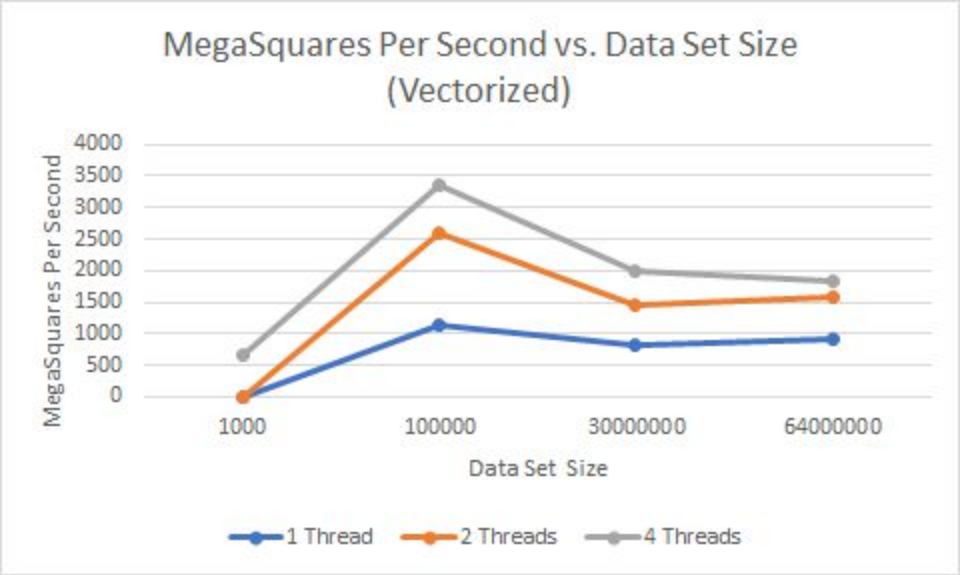
	1000	100000	30000000	64000000
1	∞	1146.68	811.12	898.93
2	∞	2597.55	1464.18	1579.91
4	662.42	3366.55	1985.28	1819.85

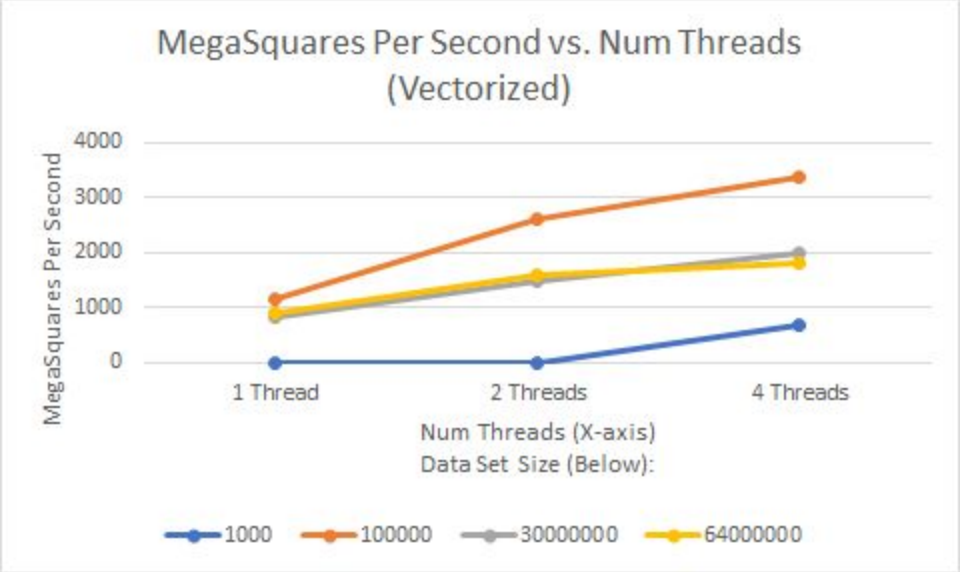
Show the graphs

Note: The third graph represents infinity at zero so the other data is not too skewed.


MegaSquares Per Second vs. Data Set Size
(Vectorization Disabled)
MegaSquares Per Second
700
600
500
400
300
200
100
0
1000
100000
30000000
64000000
Data Set Size
1 Thread
2 Threads
4 Threads


MegaSquares Per Second vs. Num Threads
(Vectorization Disabled)
MegaSquares Per Second
700
600
500
400
300
200
100
0
1 Thread
2 Threads
4 Threads
Num Threads (X-axis)
Data Set Size (Below):
1000
100000
30000000
64000000





What patterns are you seeing in the performance curves? (Pay special attention to what the curves are doing at the far end of NUMS.)

There are massive performance gains when vectorization is enabled. Additional threads tend to add performance boosts. The performance gains per thread seems to max out; a data set size of 30 million and 64 million have about the same MegaSquares per second for a single NUMT. For vectorized data, there were larger performance gains for a data set size of 100 thousand than for 30 or 64 million.

Why do you think the patterns look this way?

Vectorization is much more quick than non-vectorized methods because the simple calculations of a square root can be done massively in parallel when vectorization is enabled. Taking advantage of the GPU allows for massive performance increases. More threads tend to increase performance as well as more cores can be working on the task simultaneously. This can be seen for both the vectorized and non-vectorized case. Once a thread maxes out it's efficiency, a larger data set size will not see greater performance gains. This can be seen for data set sizes of 30 million and 64 million for any given NUMT, vectorized or non-vectorized. The performance gains begin to drop off as the data sets get very large. I assume this is because of the overhead necessary to keep track of and order data sets that are particularly large.

What does that mean for the proper use of vectorized parallel computing?

Vectorized parallel computing can be a very powerful tool if used properly. When dealing with very large data sets of simple operations, vectorized parallel computing is a necessity. The data above shows that the highest performance from the non-vectorized case was about as quick as the slowest case from the vectorized data. This truly emphasizes its importance in computing.