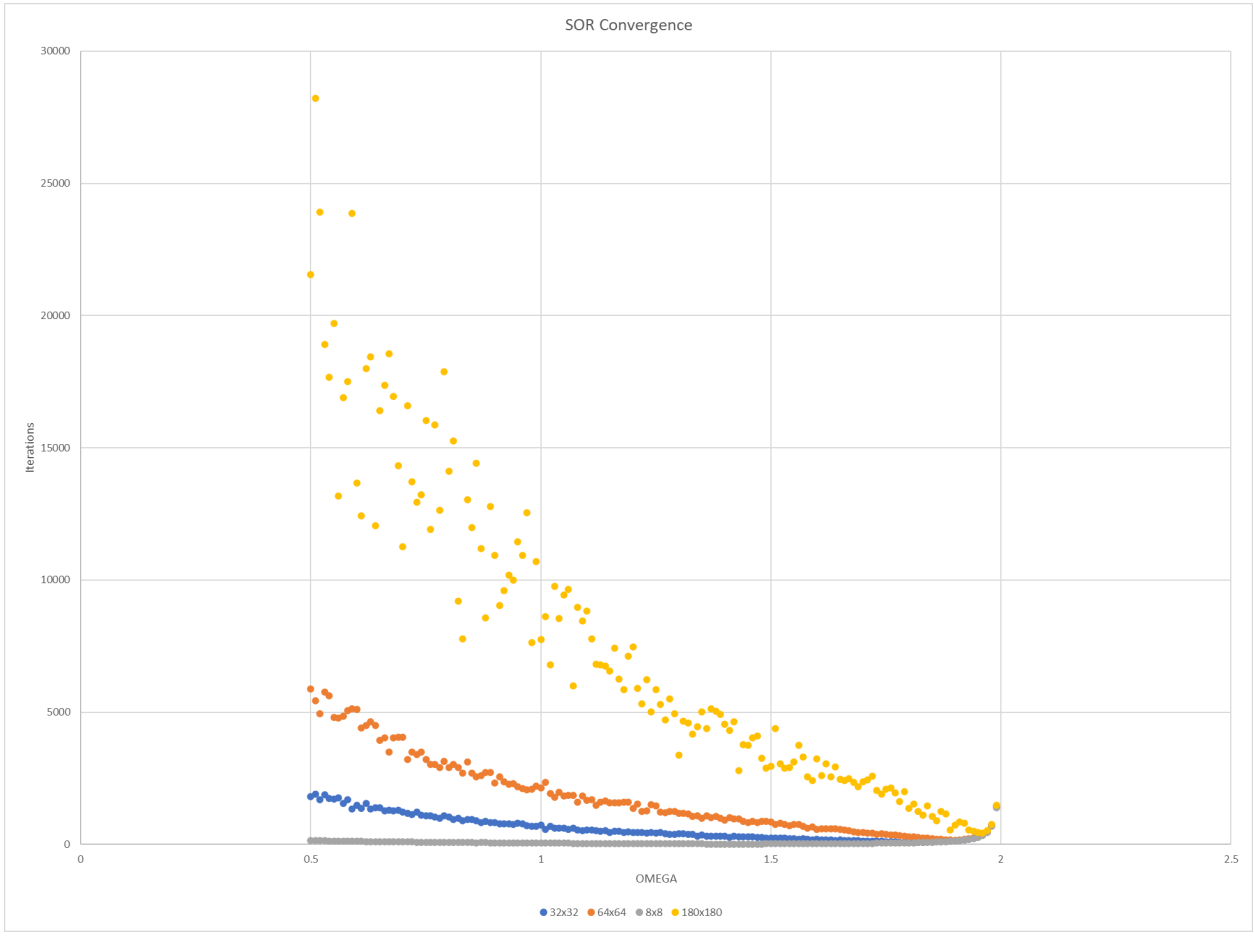
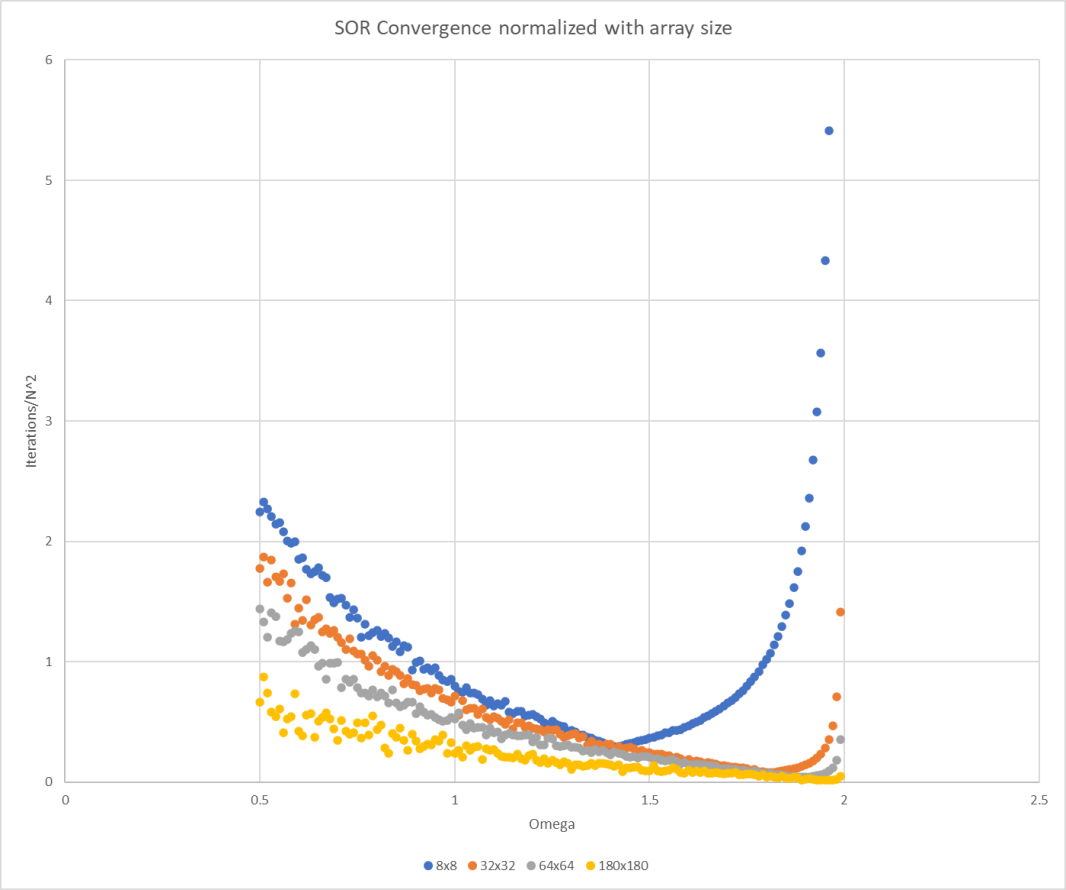
Evan Lang
Yang Lu

EC 527 Lab5

1.



The SOR Convergence over 4 array sizes. This data is not normalized for the size of each array. Data is available in Lab5Q1.xlsx



Normalized data, the y-axis is now iterations/n^2.

a. All graphs display similar behavior, the number of iterations starts high at low omegas, then as the omega increases the number of iterations decrease until we hit the optimal omega. From that point on the number of iterations increase as we move beyond that optimal omega. Finally they all experience a rapid increas in the number of iterations as we approach an omega of 1.99
b. The larger the array the larger the observed optimal omega with the 180x180 array sizes optimal omega being 1.955 and the smalles array size 8x8's optimal omega being 1.40.
c. The sensitivity to omega appears stronger at lower array sizes. Once we are passed the optimal omega the sensitivity gradually increases. As we approach 2 the sensitivity increases exponentially, shown by the extreme increase in iterations.
d. Given an optimal omega the array size itself has the largest effect on the number of iterations. This takes the form of 2 opposite observable relationships. As the array size increases the number of iterations increases. However, oppositely, as the array size increases the number of iterations normalized to the size of the array actually decreases substantially. The normalized iterations per array size of the 180x180 is actually lower than that of the 8x8, even when at the 8x8's optimal omega for both.

2.

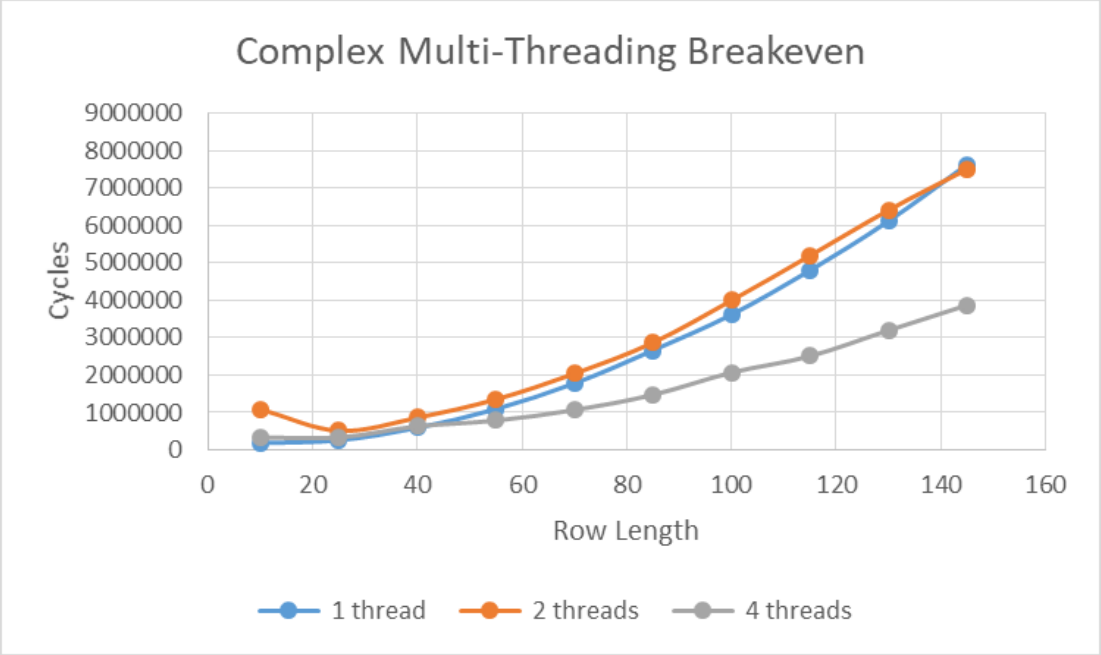
size	SOR time per innermost loop	red/black time per innermost loop	SOR_ji time per innermost loop	SOR_blocked_16 time per innermost loop	SOR_blocked_8 time per innermost loop
32	60.31436012	28.43445617	33.14499628	30.36644345	30.30831473
64	55.73444538	12.31018926	32.11219032	28.28512106	27.93320664
128	30.36107772	9.690247105	27.58018599	18.08181552	15.08423092
224	21.93858645	9.620145527	27.62345057	17.24365018	14.10318945
352	21.28713988	8.945042909	27.23051997	15.7027559	12.63134471
512	22.24656785	9.663280596	27.74890896	17.37028816	14.60161899
704	22.59529868	8.557427743	28.08569197	17.62495817	14.57284513
928	22.21532648	8.553700673	27.94424736	17.25679837	14.13320581
1184	22.34944123	8.639640744	28.05428178	17.44346633	14.37958162
1472	22.26255737	8.522217289	28.00483916	17.30535316	14.18747359

All data is available in Lab1Q2.csv. I am quite surprised by the result. Red/Black has a definitive advantage over all the other optimized forms of SOR. I am not surprised that ji leads to a regression in performance when compared to the standard strategy at high array sizes. It is the most vulnerable to cache size restrictions we see when the array sizes are greater than the l2 or l3 cache. Blocking shows moderate performance improvement, with blocks of 8 outperforming blocks of 16. However this improvement does not compare to MMM as the SOR memory access locations are not often all contained within the size of each block. That is why once we reach large array sizes, (in this case 128 appears as a good threshold) we don't see any further improvement to the performance.

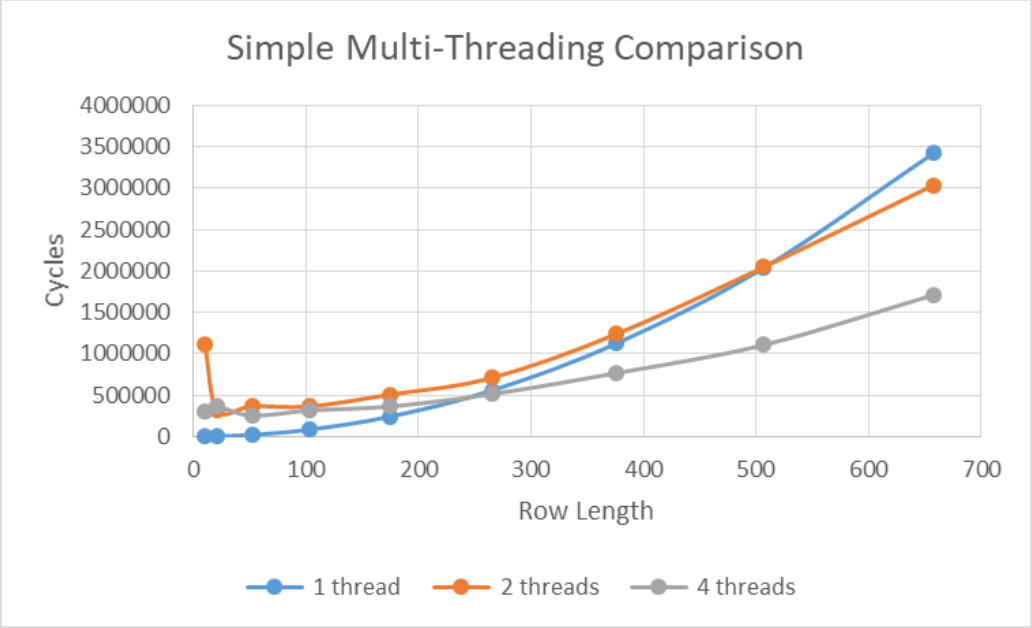
3.

The overhead for threading is shown below:

	Single Thread	Two Threads	Four Threads
Overhead (microseconds)	0.0347	36.01	59.04



For complex operations, the break even point between a single thread and quad threads is approximately 40^2 number of elements. The break even point between single thread and dual threads for complex operations is approximately 140^2 number of elements.



For simple operations, the break even point between a single thread and quad threads is approximately at 270^2 number of elements. The break even point between single thread and dual threads for simple operations is approximately 507^2 number of elements.

Although threading when possible is always a good idea, it should be noted that there exists a small overhead for multi-threading. Thus, for multi-threading to be better than the serial implementation, the size of the data and the complexity of the threaded operations need to be accounted for.

4.

Row Length	Serial SOR (sec)	Threaded ij SOR (sec)	Threaded ji SOR (sec)
1700 (in L3 cache)	24.88	1.513	2.706
1900 (out of L3 cache)	29.43	1.676	2.832

Based on the table shown above, it looks like the serial version of SOR is significantly affected by not being able to load from L3 cache. The two threaded versions, on the otherhand, see very minor differences when compared between being in L3 and being out of L3 cache. Based on this it can be concluded that threading helps significantly when data cannot fit within the L3 cache.