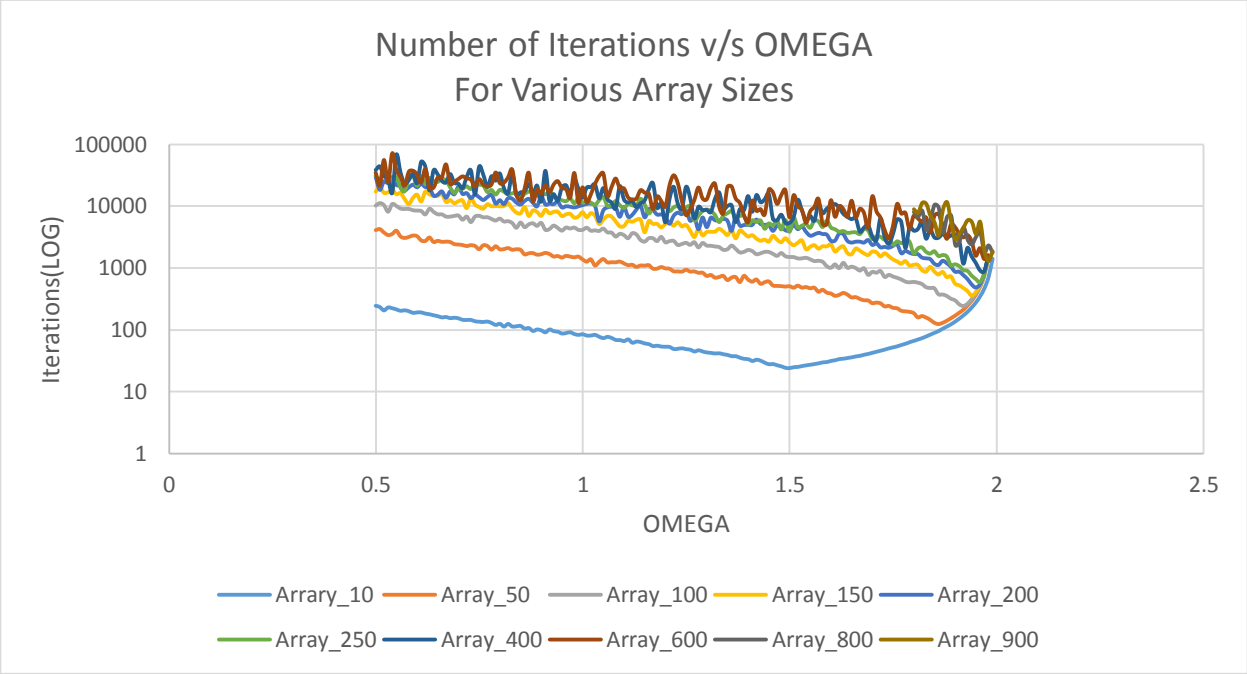
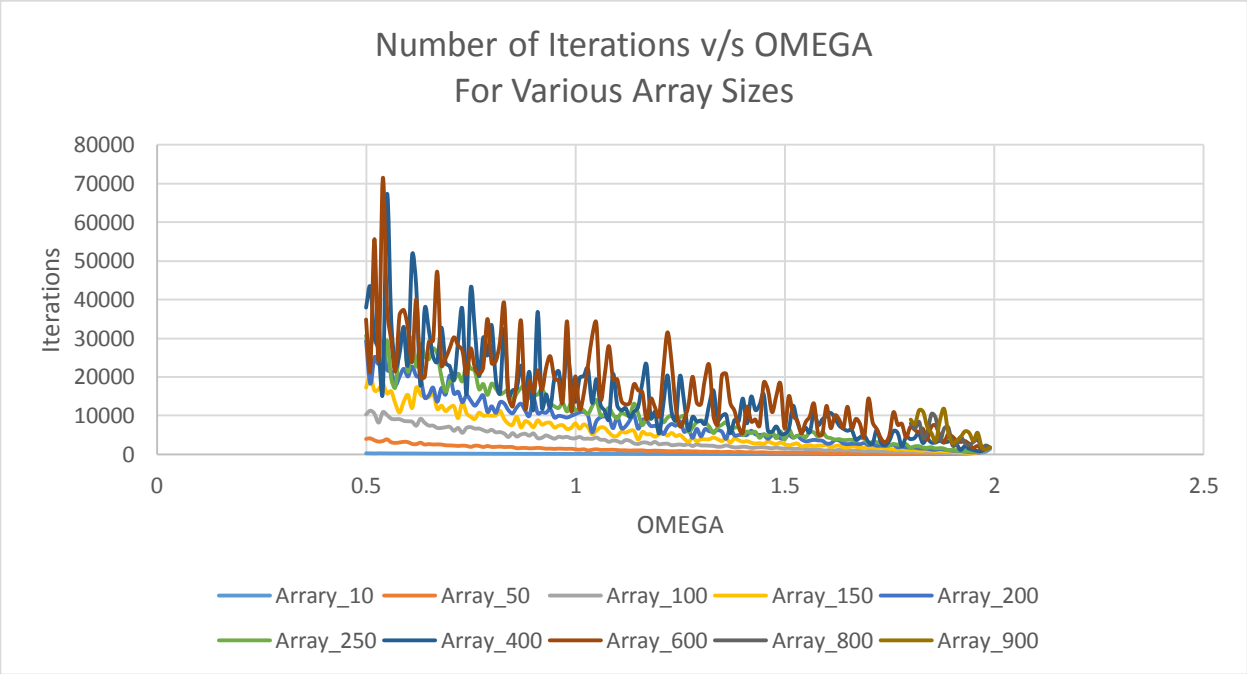
EC527 Lab 5

Part 1:



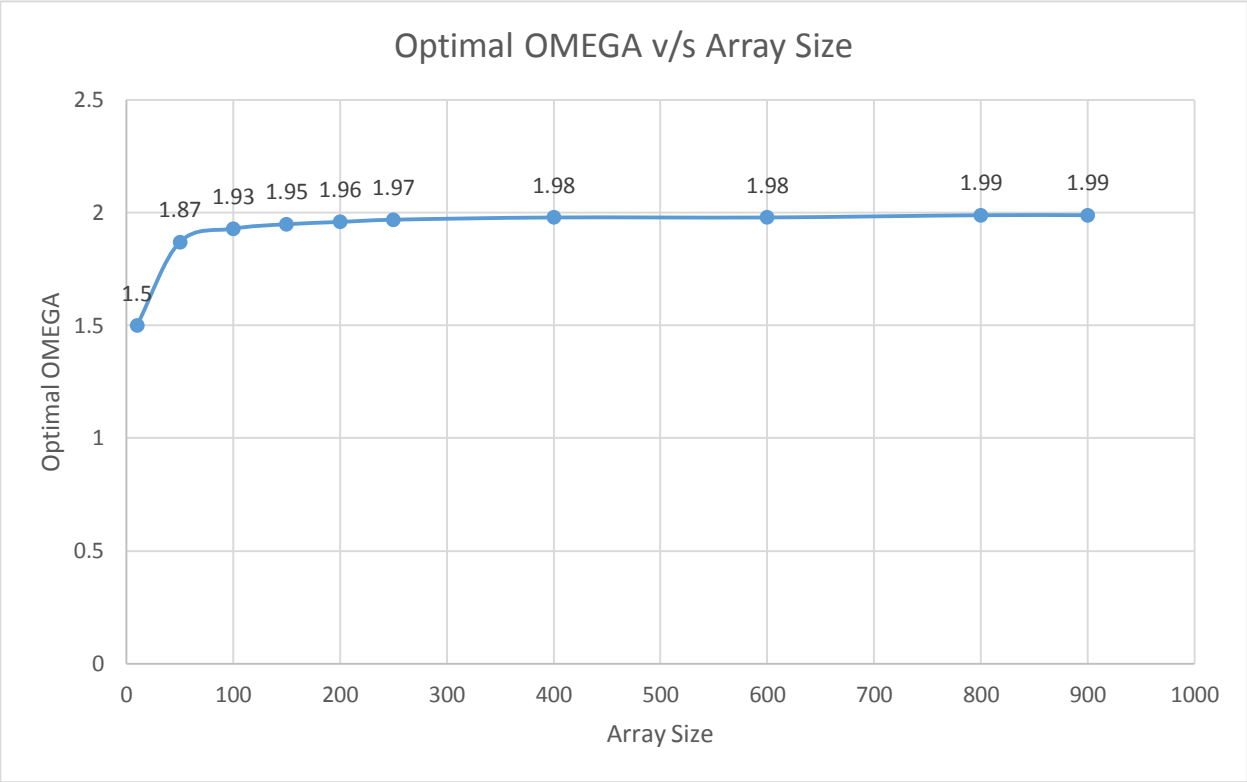
We see that when OMEGA is varies from 0.5 to 2 for a range of array sizes, the number of iterations decrease. This leads to an increase in the convergence. However, the convergence only takes place till OMEGA reaches an optimal value. Beyond the optimal value, the convergence as well as the iterations increase.



- I observed that the optimal value of OMEGA increases with the increase in array size but never goes beyond the value 2.0.

- Within the range of 1 and 2, OMEGA is found to be the most sensitive. I observed that beyond 1, the number of iterations keep on decreasing with the increasing OMEGA. This happens only till OMEGA<2.

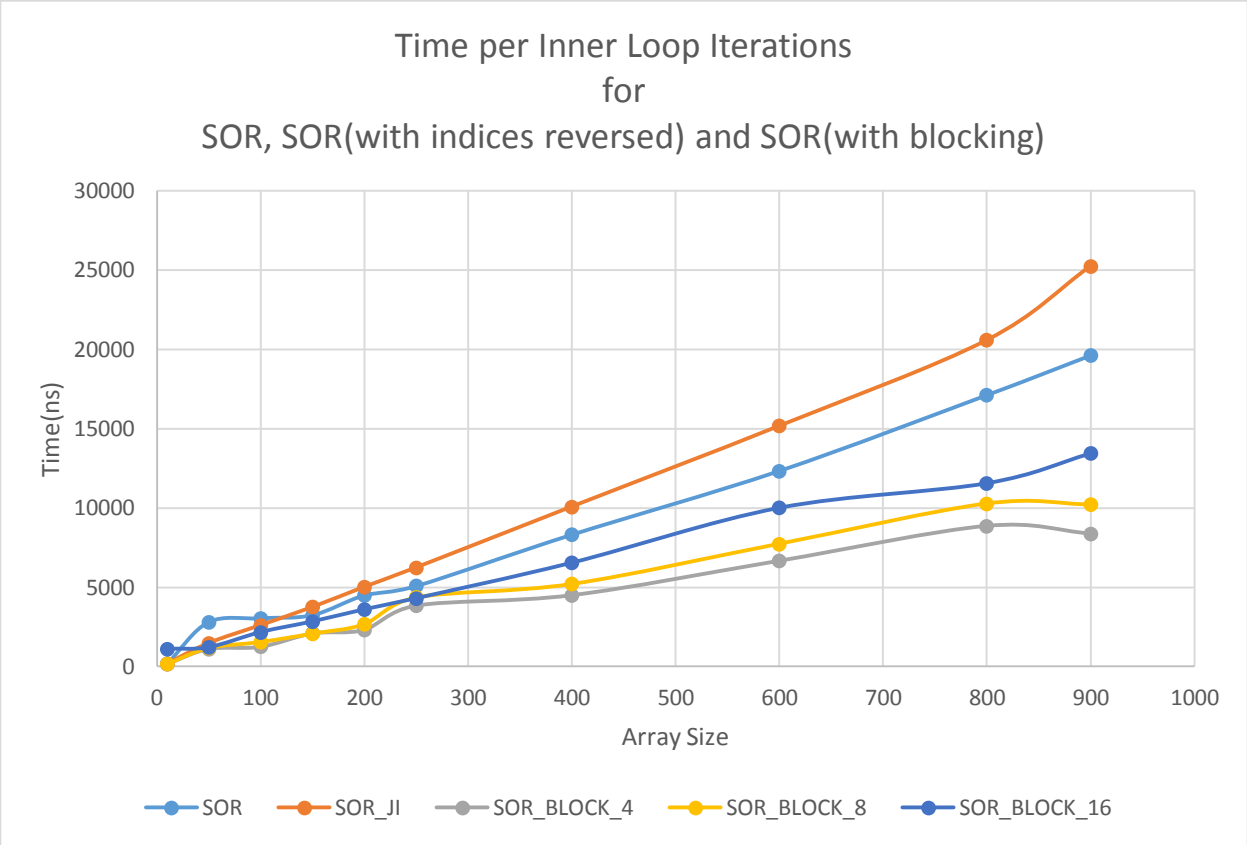
- For a given value of OMEGA, the number of iterations are also affected by the cache size. The number of misses increase if the array cannot fit in the cache and hence the number of iterations also increase.



The optimal value of OMEGA is **1.90.**

Part 2:

The block sizes used for experiments are 4, 8 and 16.



When SOR is performed with blocking having a block size of 4, the best performance is observed. This was followed by block size 8 and block size 16. The SOR with indices reversed gave even worse performance.

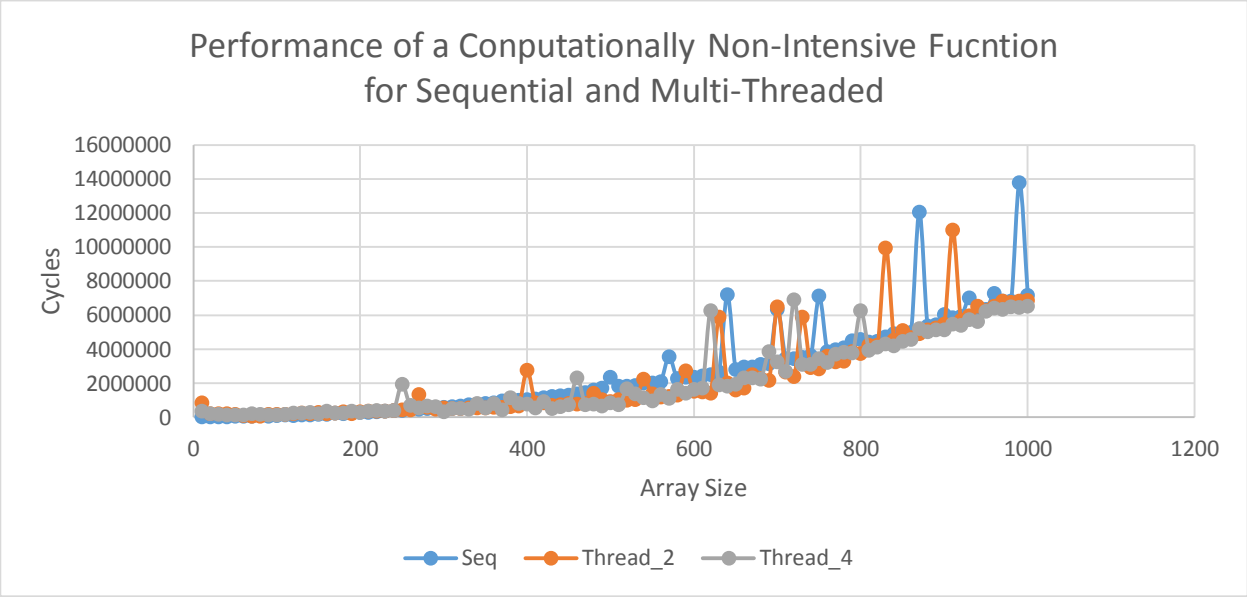
The results obtained were quite surprising since my expectation was to see the performance increase with increasing block size.

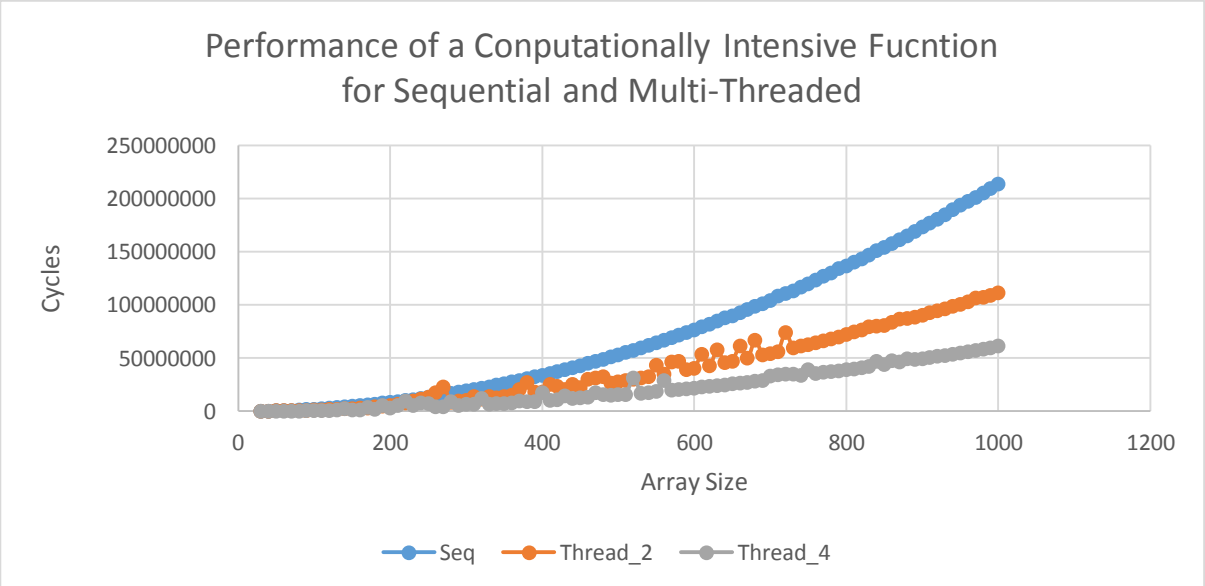
The possible reason for this could be the inability of the block to fit in the cache leading to more number of misses and hence a degradation in performance.

Part 3:

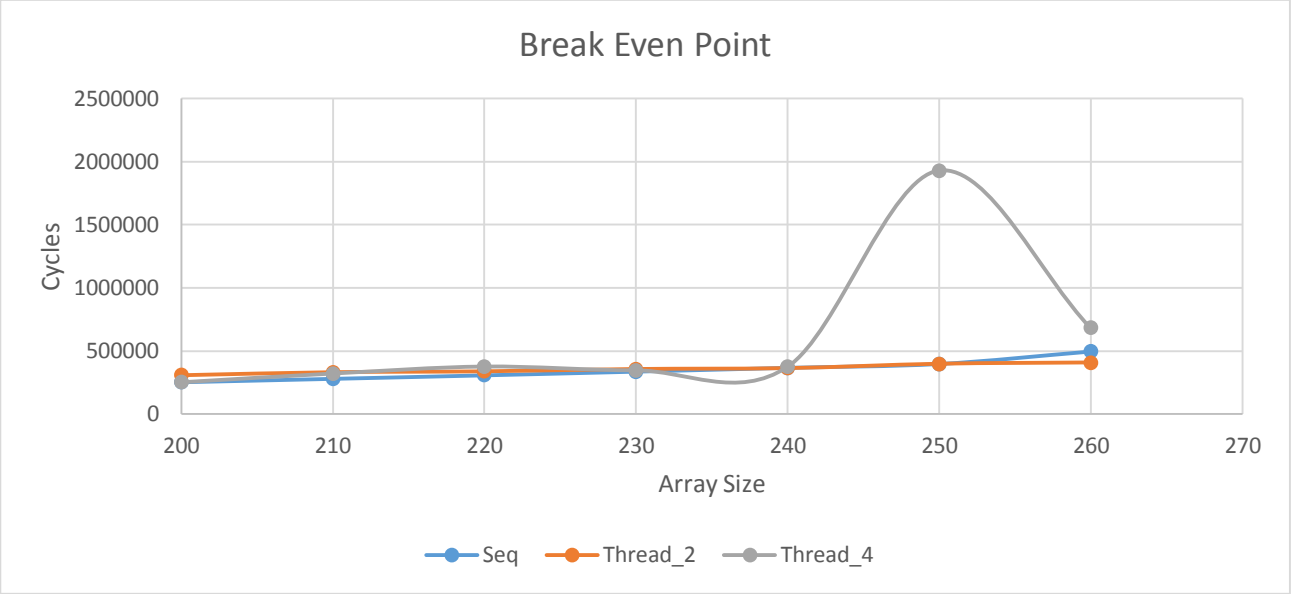
I observed that for computationally intensive functions, the multithreaded version of the program outperforms the sequential since the computations requires a lot of steps and all the steps are happening in parallel for multi-threaded system.

However, for computationally non-intensive functions, the performance observed from sequential and multi-threaded version are nearly the same, with sequential outperforming the multi-threaded one at times.

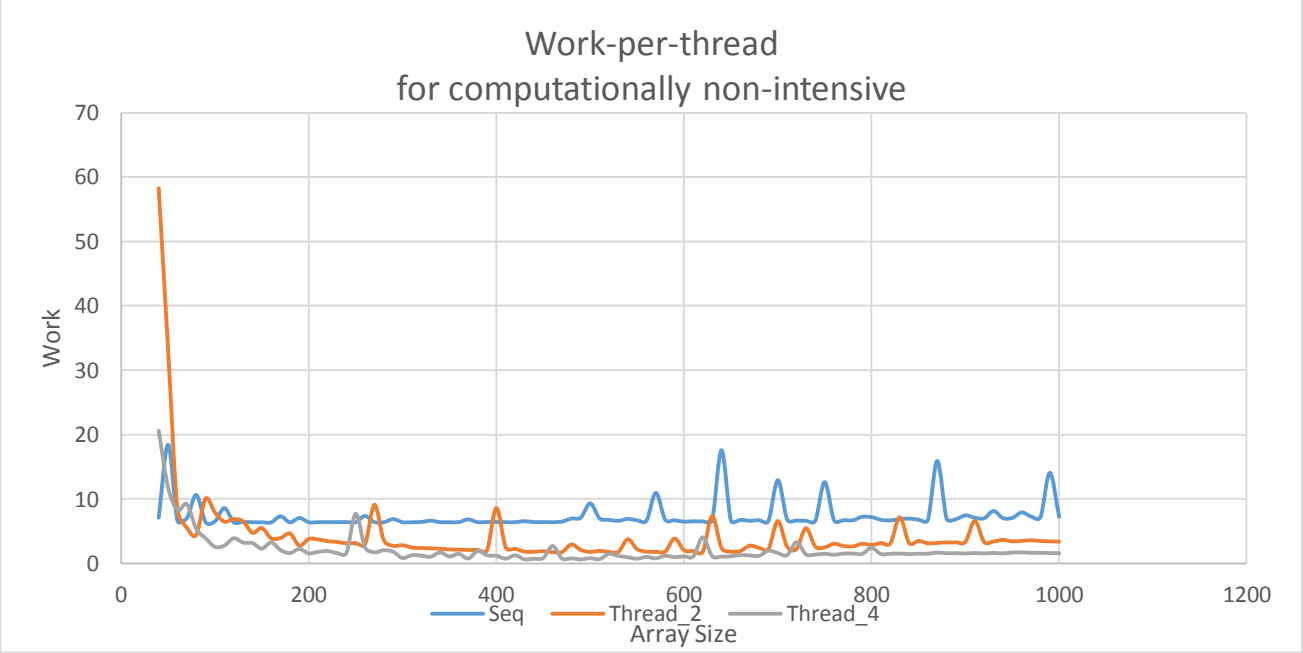


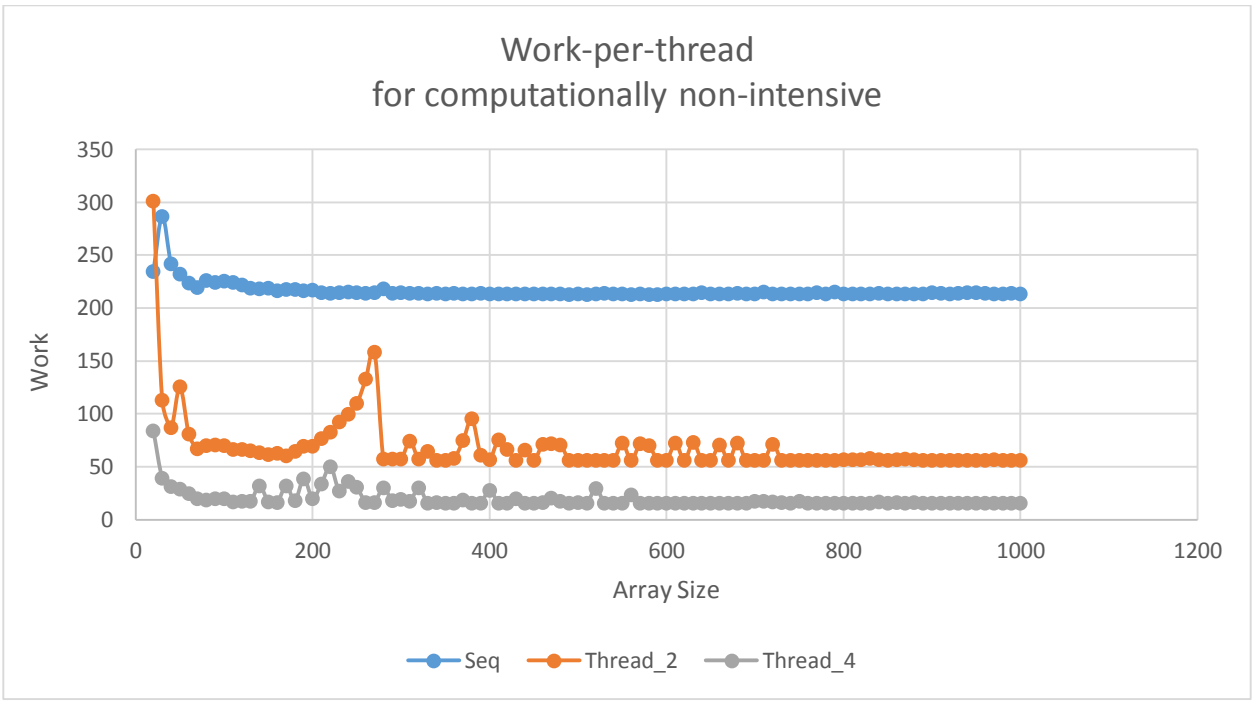


I found the bread-even point to be somewhere around the array size of 230-240.



The graphs below show that for computationally intensive functions, the work per thread is lower for multithreaded system than the sequential system.




Work-per-thread
for computationally non-intensive
Work
Array Size
0
50
100
150
200
250
300
350
0
200
400
600
800
1000
1200
Seq
Thread_2
Thread_4

Part4

I wrote the code for multithreaded SOR. The name is “test_multithreaded_SOR.c”. I tried two sizes of the arrays: one is 300 and other is 900. I chose this because one will fit in the cache and the larger one won't fit in cache. The performance is shown below:

	Non Blocking		Blocking		
Array Size	Cycles	Iterations	Cycles	Iterations	
900	6602276634	4178	11487821385	7685	B=4
900	6590689821	4178	11884422982	7886	B=8
900	6587768383	4178	12164220904	8103	B=16

	Non Blocking		Blocking		
Array Size	Cycles	Iterations	Cycles	Iterations	
300	74475719	732	80004726	775	B=4
300	71959522	732	83754566	775	B=8
300	71887657	732	82872696	752	B=16

From the above observations I can conclude that for larger arrays the number of iterations taken by the non blocking code it less than the number of iterations taken by the blocking code. So the multi-threaded version bithout blocking works better than the blocking code.