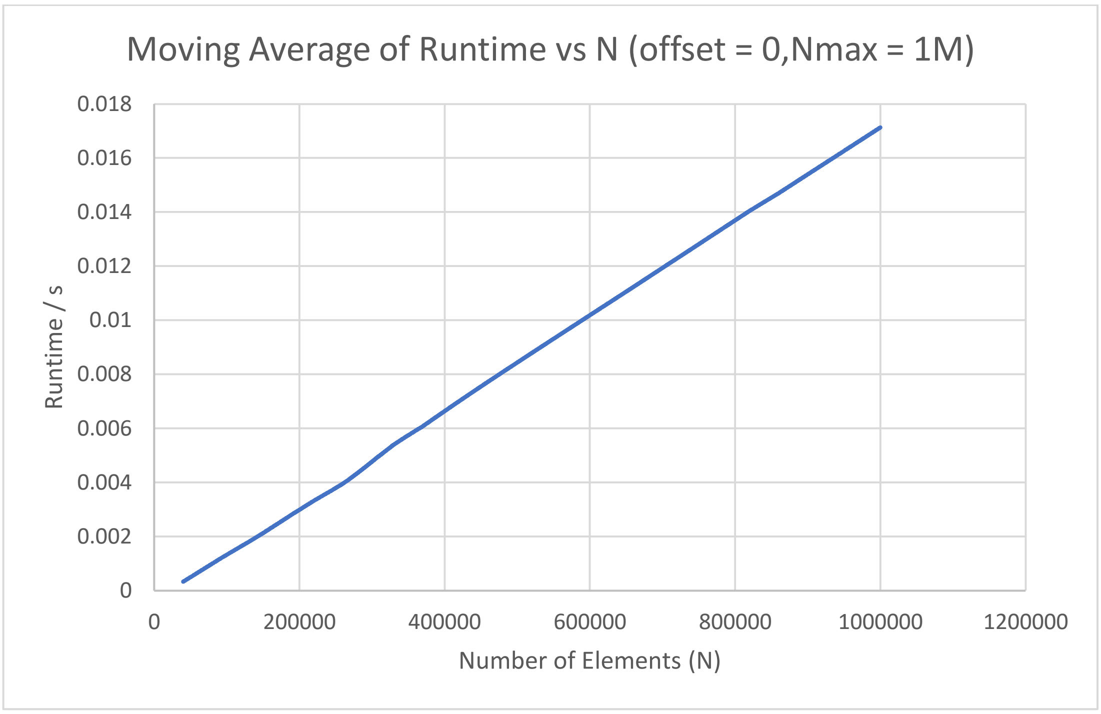


Figure 1

The plot above shows the first test checking the runtime of the loops at 0 offset varying N between 0 and 1,000,000. Due to the cache sizes L2(256kB) and L3(8MB) and the fact that each N corresponds to 24 bytes of memory, runtime jumps were expected at:

$$N = \frac{256kB}{24bytes} = 10{,}666.67 \; and \; N = \frac{8MB}{24 \; bytes} = 333{,}333.33$$

This however was not the case. Apart from a kink in the graph around 335,000 points, the graph showed a general linear trend with a gradient of around 1.5E-8 per increment in N. This inconsistency between theory and practice was explained by the fact that the optimizer and CPU used pipelining and other techniques to improve time efficiency to and minimized the expected effects.

As can be seen from the graphs, at N= 10,000, for the tests run, there was no considerable jump in runtime. This was further confirmed by testing a “zoomed in version of the 0-20000 range as shown in Fig. 2 below. Again the resulting plot was explained away by the fact that there was efficient pipelining so that a considerable time efficiency was created to prevent the runtime.

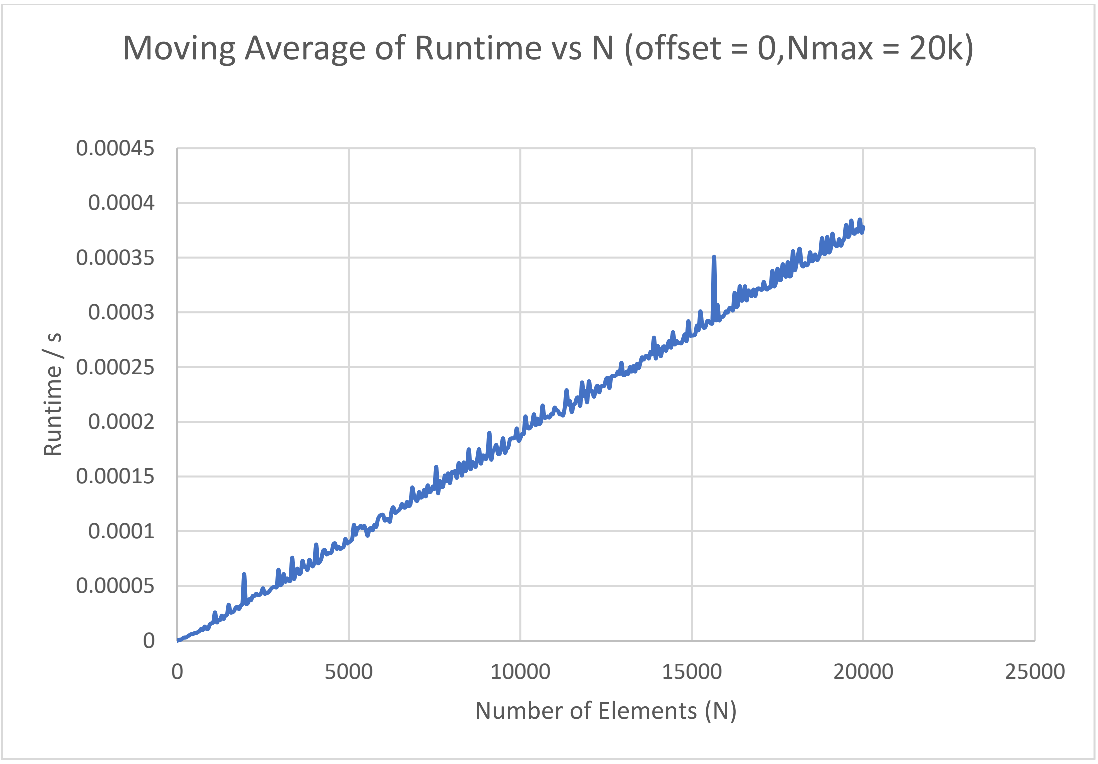


Figure 2

At around 330,000 elements however, I was clear that there was a kink in the graph shown in Figure 1. This range was investigated with another test shown in Fig. 3 below (N between 0 and 400,000). As observed, at around 330,000-370,000, there's a significant increase in the average runtime which indicated the jump from the L3 cache into main memory. Because of the switch from the L3 cache to main memory, the time taken to access data increase meaning an increase in the average runtime for those operations. This confirmed the importance of memory access in runtime speed and optimization.

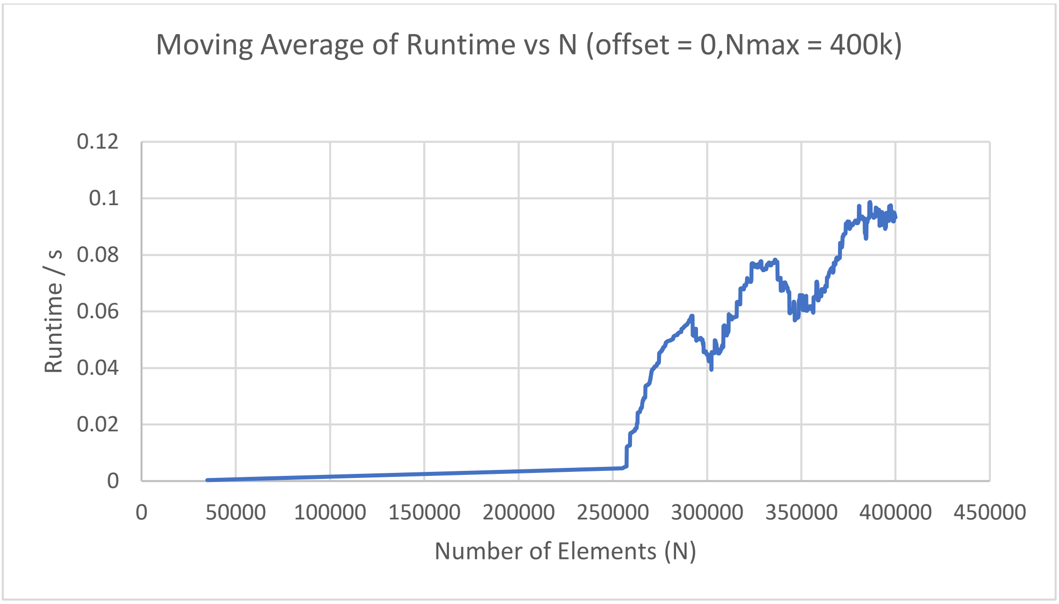


Figure 3

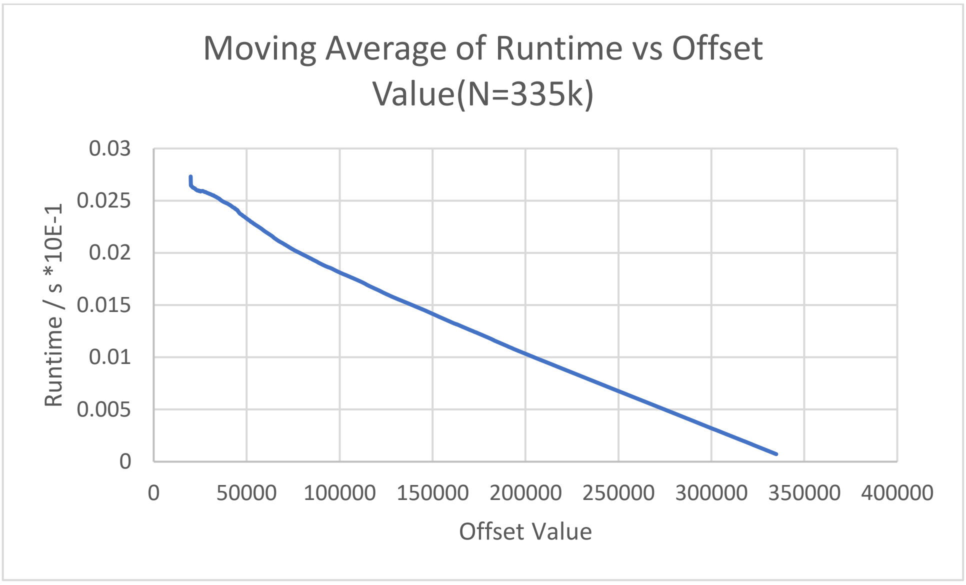


Figure 4

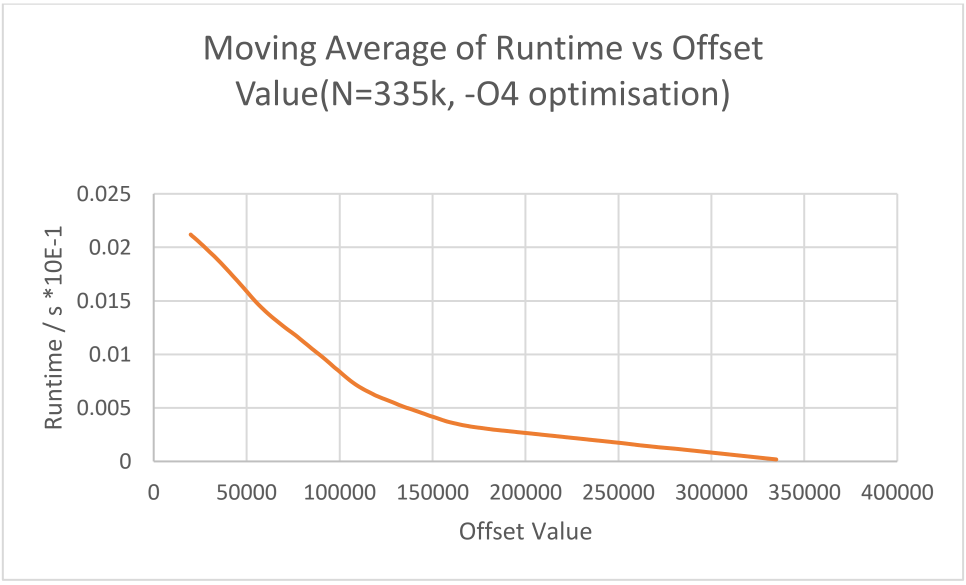
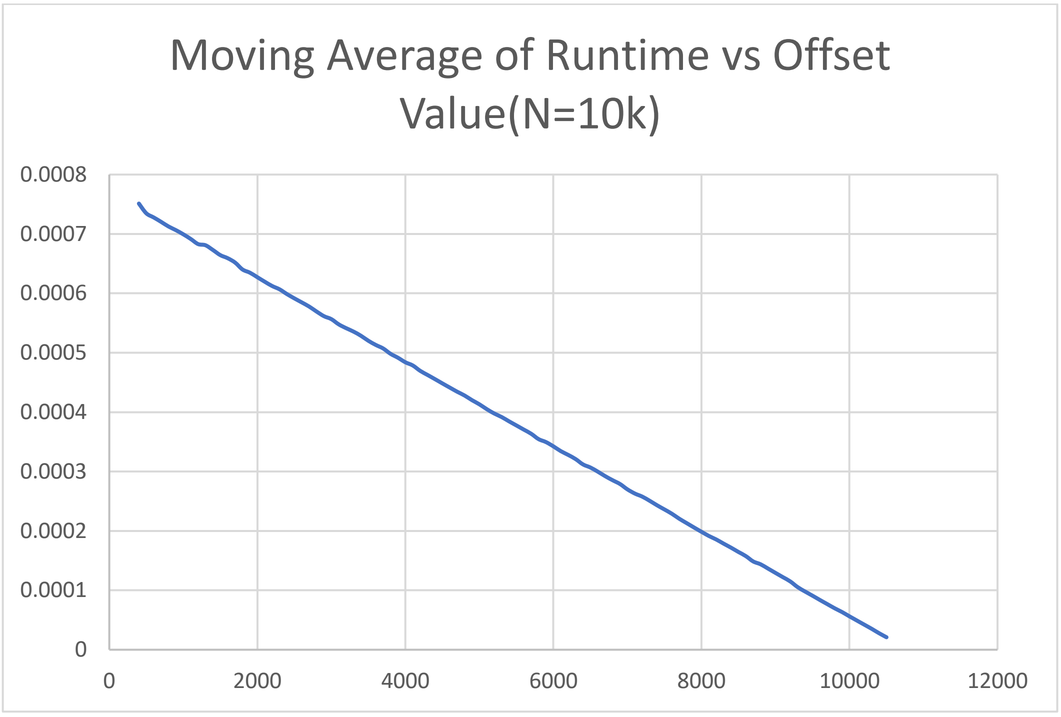


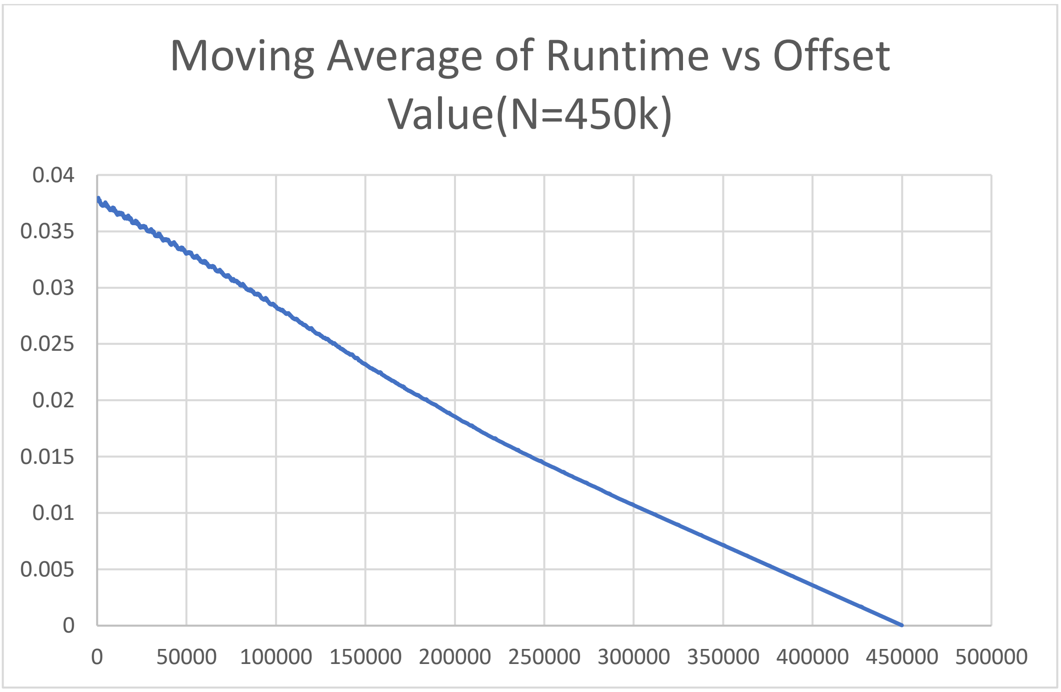
Figure 5

To test the relationship between runtime and offset, N was set to 335,000 and the calculation was performed. Figure 4 and 5 show the unoptimized and optimized code respectively.

As the offset increased, it can be seen in both cases that the run time decreased, this was expected because the number of times the operation was to be performed decreased. It could clearly be seen that the optimized curve had a generally lower average runtime especially at offsets of between 50000 and 200000. This was again expected as the -04 optimizations mean more efficient code and perhaps pipelining.

The test was repeated for values of N = 10000 and N = 450000 and produced similar relationships between the runtime and offset. The plots are shown below.


Moving Average of Runtime vs Offset Value(N=10k)
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
0
2000
4000
6000
8000
10000
12000


Moving Average of Runtime vs Offset Value(N=450k)
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
50000
100000
150000
200000
250000
300000
350000
400000
450000
500000