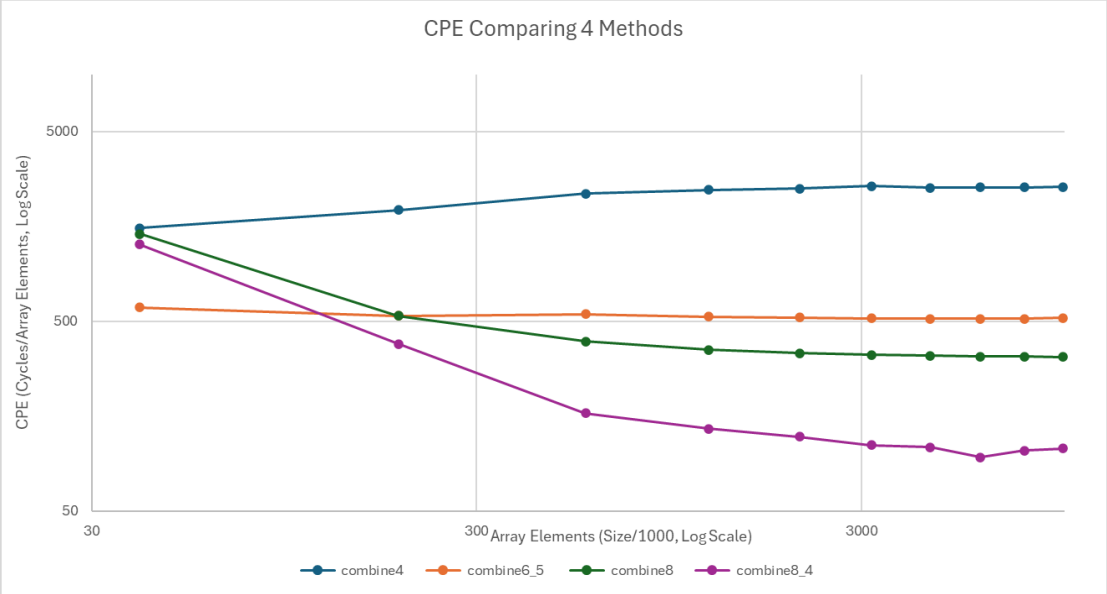
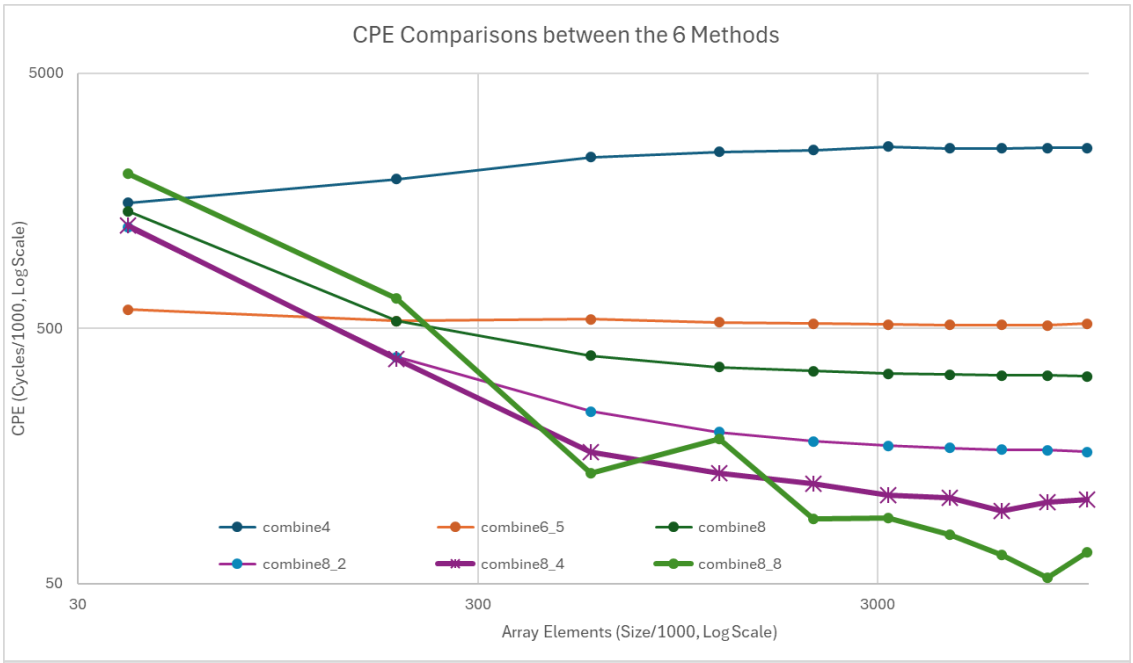
Lab 3
Chih Han “Josh” Yeh
ENGEC 527 Spring 2025

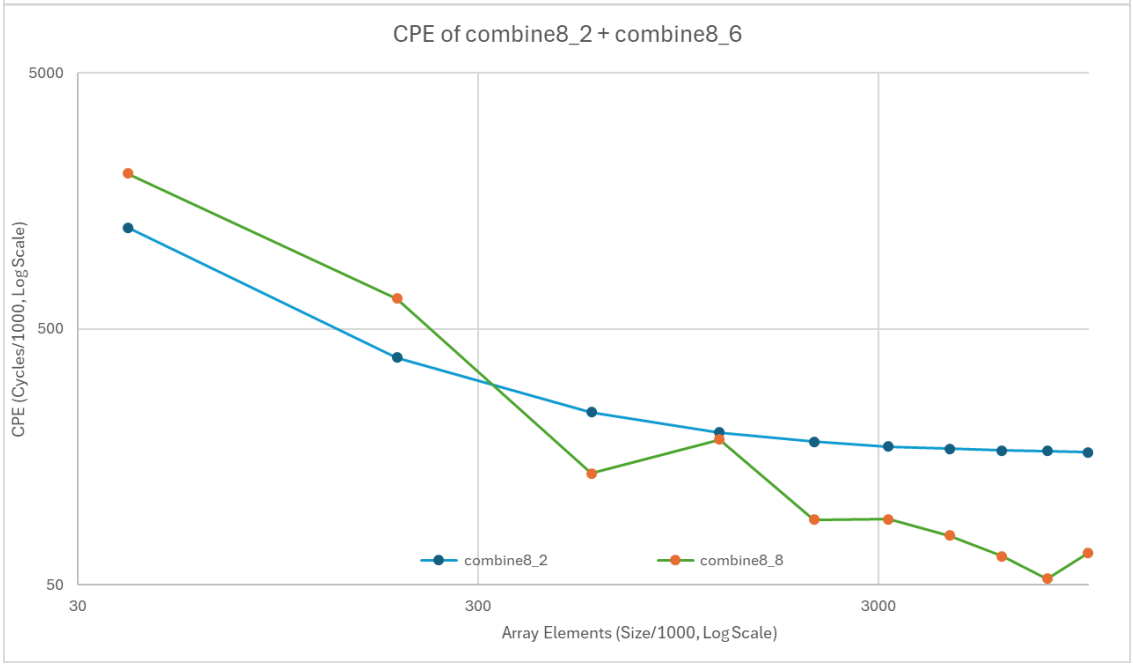
Problem 1
1a. Below is the performance between four different program execution methods, with two being scalar and two being vector. The equation I have chosen to fit the criteria (multiples of 8 which is the result of 32 bytes divided by the 4 bytes of float data types) is: $120x^2 + 28x + 40$, which would ensure the number of elements reaches ~10000 (10012) and still be multiples of VSIZE.



As expected the vector methods (combine8 and combine8_4) are faster than their scalar counterparts. The fairest comparison between scalar and vector loop unrolling would be the CPE comparisons between combine4 and combine8 which are the base methods of each. Both types of methods would start their CPEs in the 1500s (1552 and 1442 for combine4 and combine8 respectively). The CPEs substantially differ significantly when matrix size increases with combine8 dropping towards the 300s, while combine4 increases towards the 2500s (2548 and 324 respectively in this program execution). As the matrix size exceeds 1200, the CPEs of each scalar and base vector function reach a plateau. The speedup comparing the base method of vector against scalar goes from 1.0 (starting matrix) to 7.8. SIMD allows multiple data processing in parallel which increases overall performance in CPE.

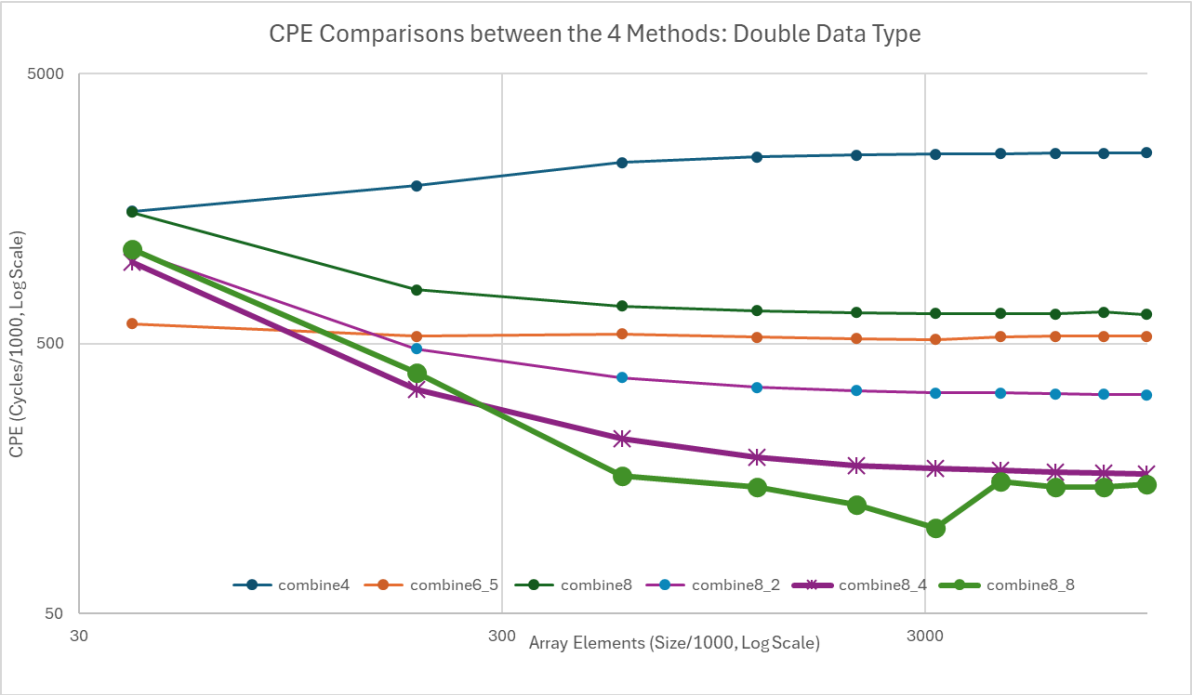
1b. Below are 2 plots comparing the CPEs of all 6 methods and vector unrolling with 2 and 8 accumulators.





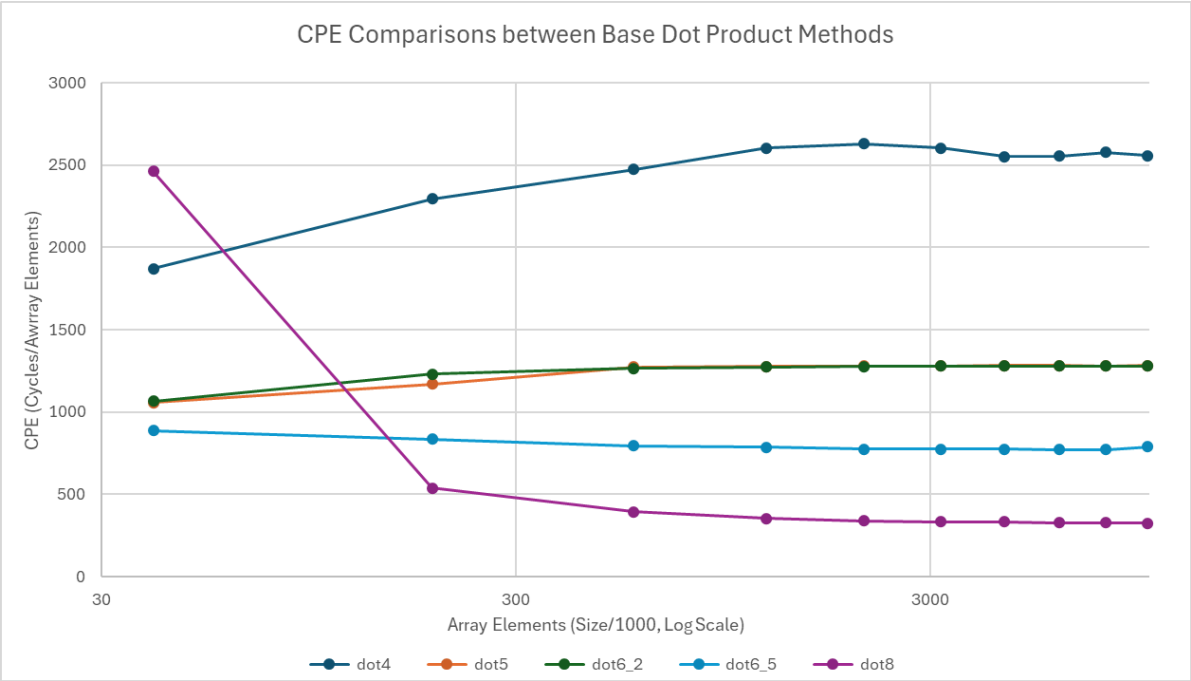
Increasing the accumulators reduces the loop dependence of previous iterated data, therefore the function allows the processor to execute iterations in parallel. This is why when comparing the CPEs of the different accumulator vector functions you can see the downward slope being steeper for higher accumulators.

1c. The magnitudes of each function increased but the differences in performance are still distinct as seen from the plot below.



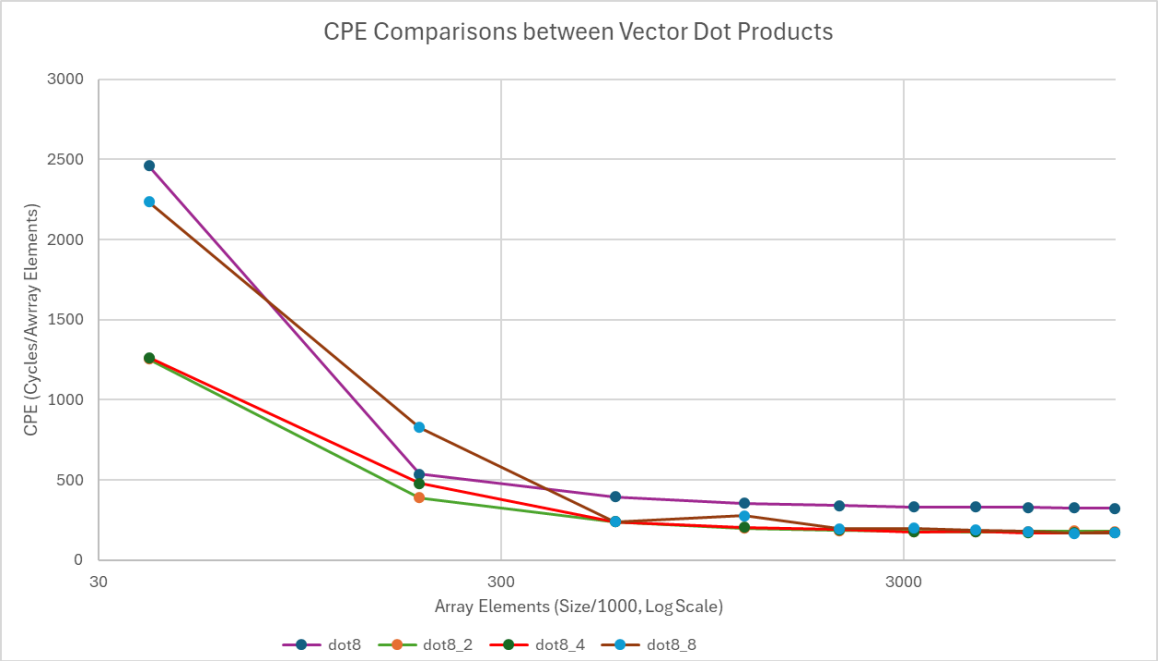
The difference in CPE between Double and Float data types is that of combine8_8 in which the CPE after 4000 would jump towards the same value as matrix size 1200s and maintain that plateau to the end. Other than this anomaly, having more accumulators for both vector and scalar methods generally helps.

1d. The performance between the scalar dot product functions and the base vector function is very similar to



the functions from test_combine8. The CPEs of the scalar functions would increase and reach a plateau when the matrix size exceeds 1200.

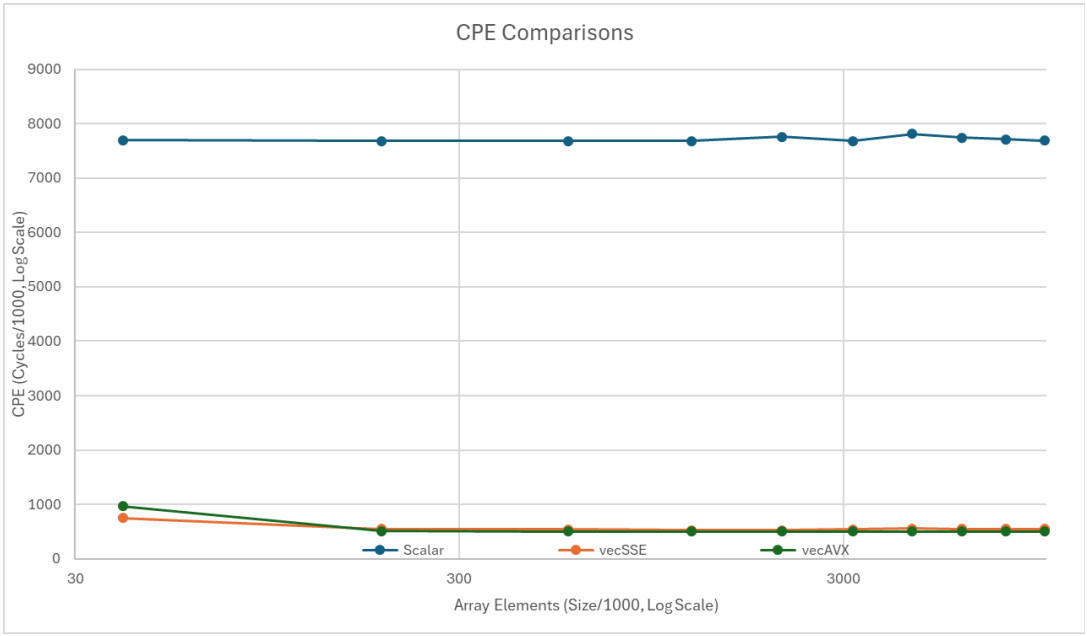
1f. Below is the CPE plot comparing the base vectorized dot product and using 2-8 accumulators.



Aside from the base vector dot product, the three dot product functions with accumulators seem to reach very similar performances, when matrix sizes exceed 2000 they all seem to plateau around the same CPE levels of 170. I think the reason for these performances from the vectorized dot products is that the basic vectorized function in itself was already making the processor execute very efficiently, and adding more accumulators seems to just be having diminishing returns in performance.

Problem 2

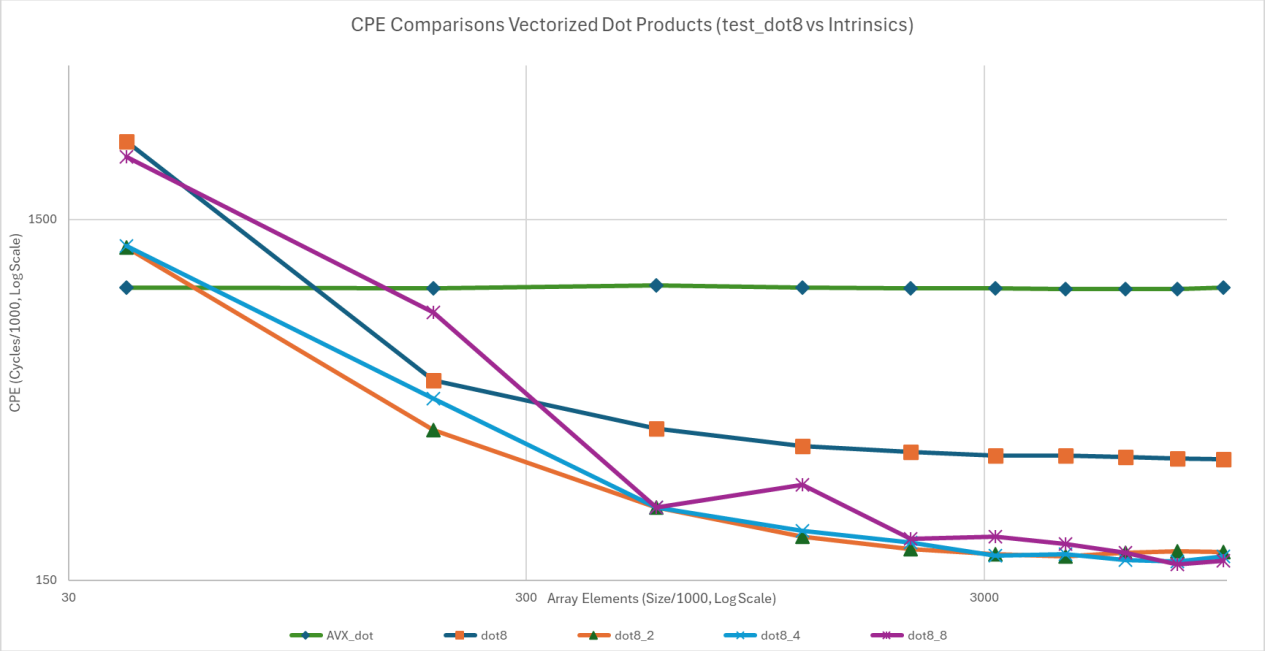
2b. Below is the plot from executing test_intrinsics.c



The performance comparisons between the methods were not what I expected, at least between the base scalar function and the two different vector functions. Both using SSE and AVX would have very comparable CPEs, very minor differences, but are more than 15x faster than the scalar function.

2c. The CPE of both element-wise add and element-wise multiply starts from 1670 and gradually reduces to the 1250s, with both averaging around +/- 1320s (1337 for add and 1316 for multiply respectively). Due to the lack of iteration dependence, both functions are vectorizable.

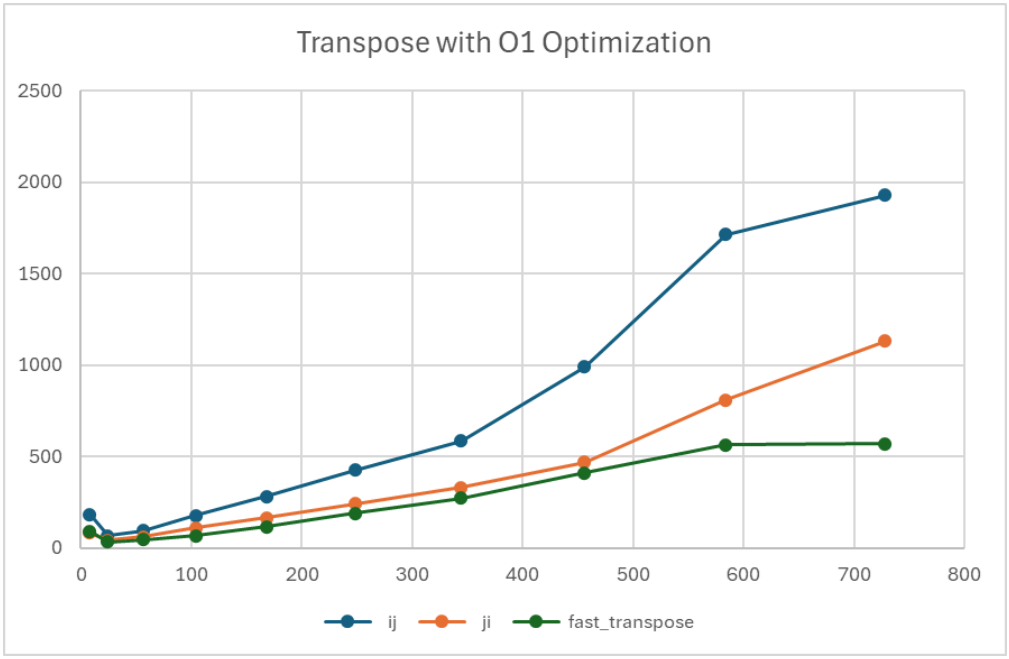
2d. Below is the plot comparing my intrinsics vectorized dot products and the vector dot product functions from test_dot.c.

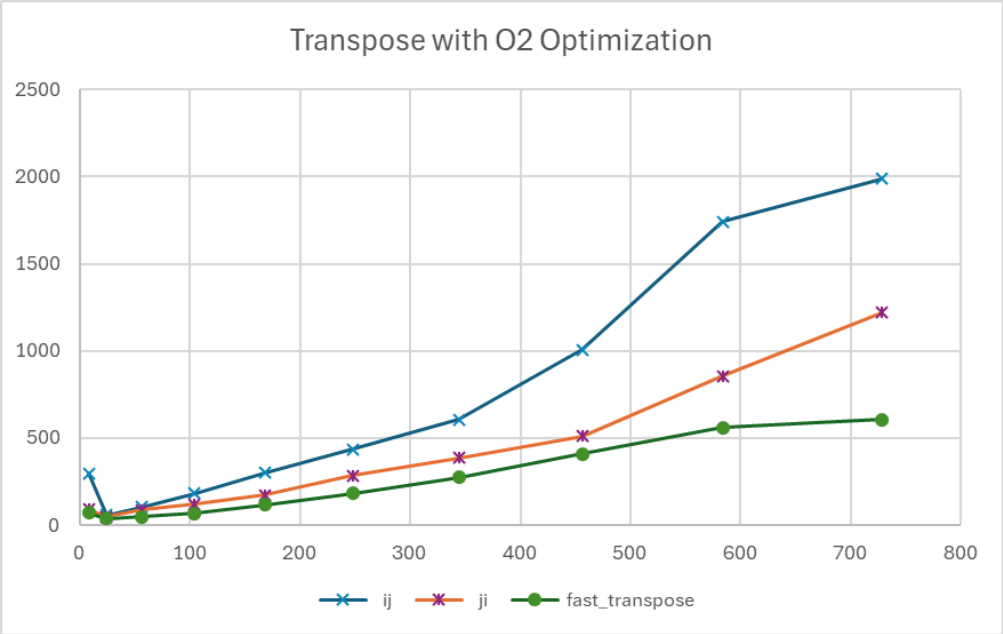


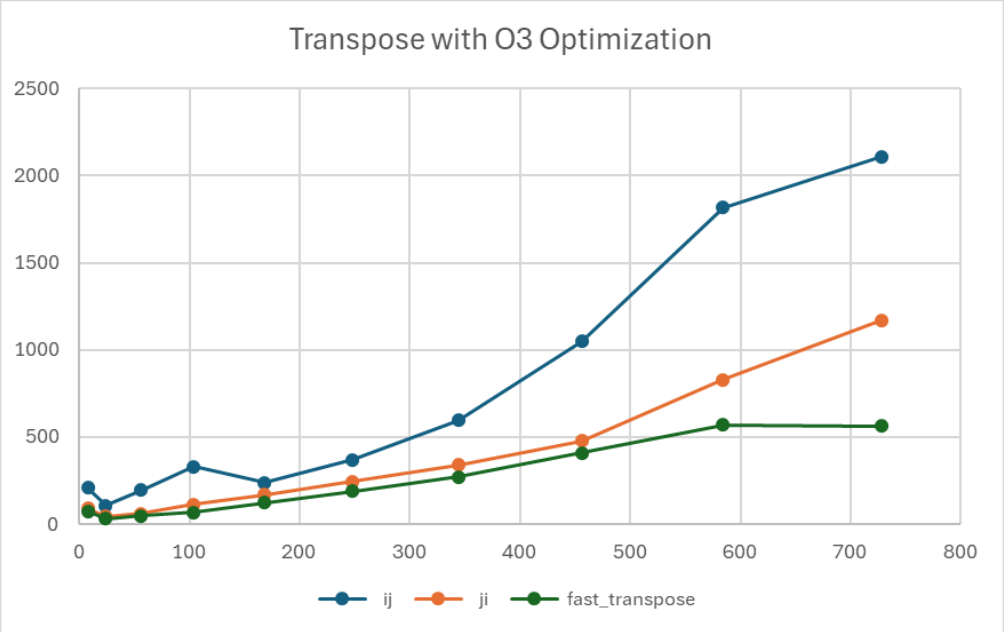
The intrinsics vectorized dot product is still significantly slower than the vectorized dot product functions. The CPE of the intrinsics function in mind plateaus around the 960s and 970s while the vectorized dot product functions with accumulators aggressively reduce and does not reach a plateau until the matrix exceeds 2000.

Problem 3

Below are several different plots of transpose methods while also modifying optimization methods (CPE vs Array Size, Y vs X-axis)







CPEs of all 3 methods generally trend towards the same pattern, with my fast transpose method plateaus as CPE reaches 600, at the 560-570 range. CPEs of the other two methods reach over 2000 and 1990 respectively. Using blocking and intrinsics, my transpose method proves to be 2-3 times the speed up compared to ij (stopping from 4.1, dropping to 1.5, and gradually rising up to 3; whilst fast_transpooor increases from 70 then drops to 40 before increasing till 600).

Part 4

a. This lab assignment took me over 6 hours to complete, most of the time was spent on understanding the types of data for collection and plotting.
b. The only problem in the lab is the missing reference to certain readings and notes that are not available.