ME 459 - HW 8

Jackson Fox

Due 11/18

Task 1

Plot of Performance

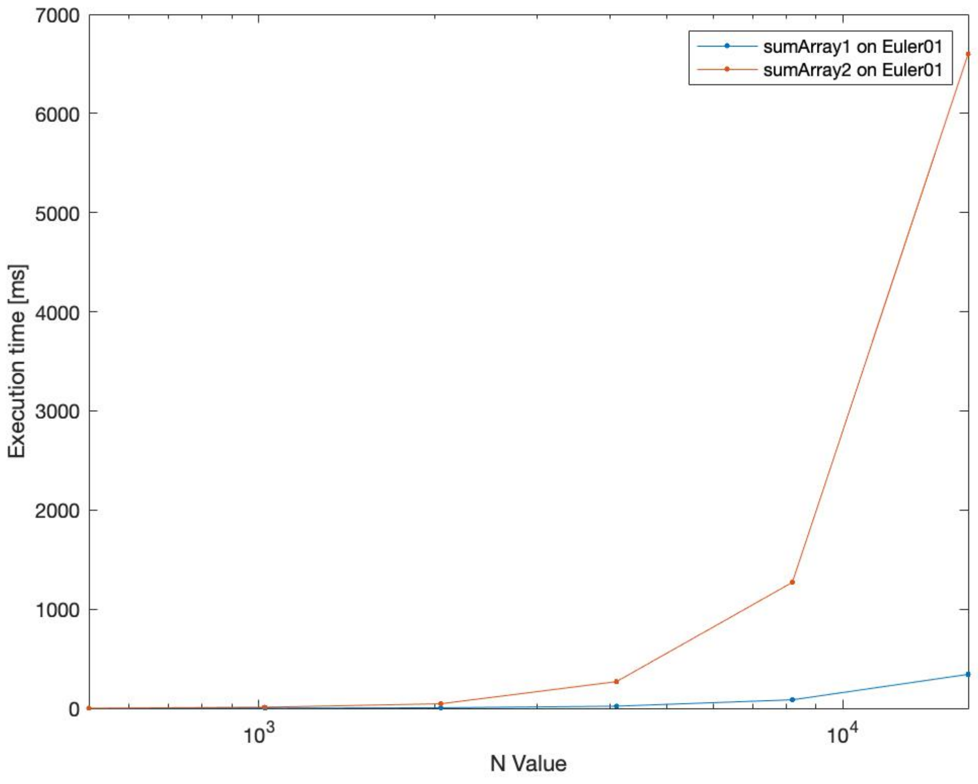

Figure 1: Plot of performance of sumArray1 and sumArray2 in milliseconds

Part e

The performance difference is due to the way that the functions parse through the arrays. In sumArray1, the entries in the array are accessed in row order, which is the same as how the entries are stored in memory. This means that a substantial number of consecutive entries are stored in the cache at once, minimizing hits to main memory, allowing for quick access. In sumArray2, the entries are accessed in column order, which is not the same as how they are stored in memory. This means that it is unlikely that when the program looks for the next entry to add that it's in the cache - to be able to access the next column the program needs to go to main memory which is significantly slower. As the number of entries in the array increase, the number of hits to the main memory also increases, which causes the dramatic slow down of the sumArray2 vs. sumArray1 with increasing n values.

Task 2

Part f

Of the four implementations, the fastest was mmul3 which took about 1.23 seconds, and the slowest by far was mmul4 which took about 20.13 seconds. Both mmul1 and mmul2 feel around 9 seconds to complete. The reason mmul3 is so much more efficient is due to the structure of the B Array. By utilizing the three for-loop approach and having B column major order, the k index sweeps through both the A and B Arrays row by row, which matches the way that the arrays are stored in memory. This means that accessing the next element in memory for both arrays can be typically be done in the cache, saving significant time from going to the main memory. The exact opposite is true in mmul4. By flipping array A to be in column major order, the k index now sweeps through both array A and B by column. This is not aligned with how the arrays are stored in memory, and will result in many cache misses when trying to locate subsequent steps. This requires going to the main memory to find the subsequent steps which is significantly less efficient.