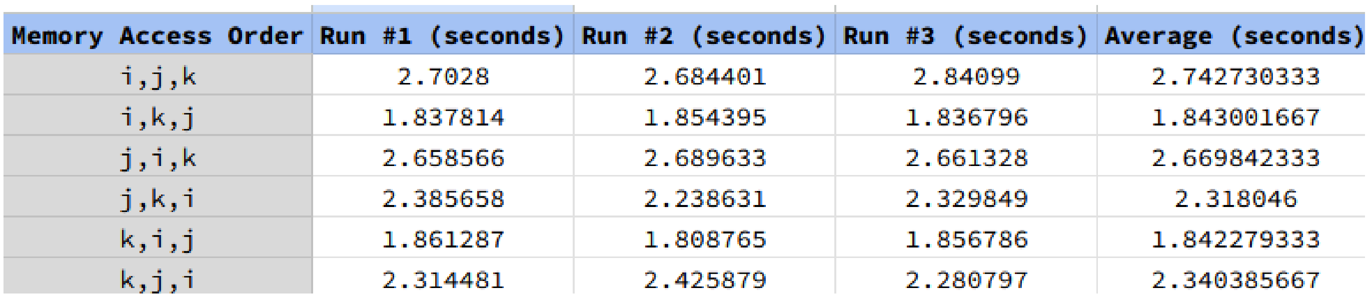
Homework 1: Locality

Author: Ashwin Pai

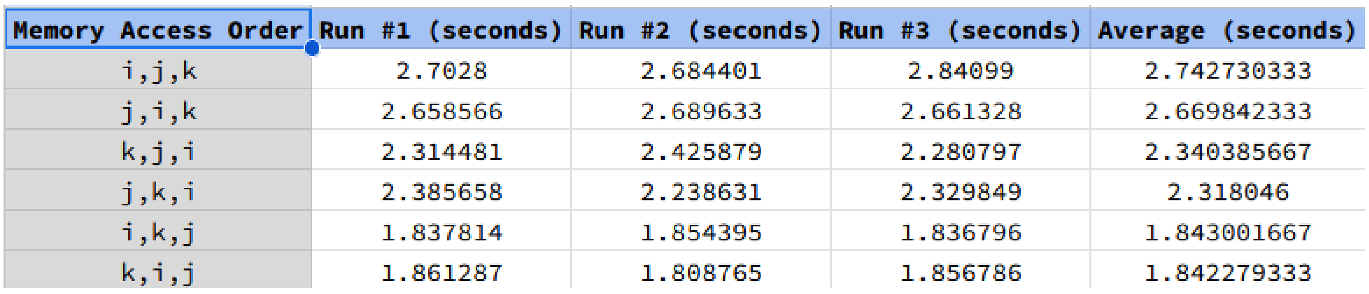


Memory Access Order	Run #1 (seconds)	Run #2 (seconds)	Run #3 (seconds)	Average (seconds)
i,j,k	2.7028	2.684401	2.84099	2.742730333
i,k,j	1.837814	1.854395	1.836796	1.843001667
j,i,k	2.658566	2.689633	2.661328	2.669842333
j,k,i	2.385658	2.238631	2.329849	2.318046
k,i,j	1.861287	1.808765	1.856786	1.842279333
k,j,i	2.314481	2.425879	2.280797	2.340385667

Results from running HW1 Benchmark

Explanation

The C programming language uses row-major ordering. This means that row elements are stored contiguously in memory. This allows for better cache locality and fewer cache misses when row elements are accessed.



Memory Access Order	Run #1 (seconds)	Run #2 (seconds)	Run #3 (seconds)	Average (seconds)
i,j,k	2.7028	2.684401	2.84099	2.742730333
j,i,k	2.658566	2.689633	2.661328	2.669842333
k,j,i	2.314481	2.425879	2.280797	2.340385667
j,k,i	2.385658	2.238631	2.329849	2.318046
i,k,j	1.837814	1.854395	1.836796	1.843001667
k,i,j	1.861287	1.808765	1.856786	1.842279333

The above image is the same data as initially shared but ranked from fastest to slowest.

Worst Case

The worst Memory Access Order (MAO) was I, J, K at 2.74 seconds. In this example, K is the innermost loop, which causes the program to jump across columns. This leads to a new row of B being fetched for every K iteration, which results in frequent cache misses and increased processing time. Spatial locality is poor for both A and B.

Best Case(s)

The best MAO cases were I, K, J and K, I, J at 1.84 seconds respectively. In both of these versions, memory is accessed in a row-major fashion, leading to better cache reuse. Both of these examples maximize cache reuse for A, B, and C, resulting in fewer cache evictions and increased performance.

Remaining Examples

- J, I, K MA: C is accessed column-wise, which is bad for cache hits and program performance.
- J, K, I & K, J, I MAO: C is still accessed column-wise, which negatively affects performance. However, these cases are slightly better than J, I, K because A is accessed row-wise, benefiting from cache locality. These cases are faster than J, I, K** because B stays in the cache longer.