**Assumptions: Cache Size: 128 bytes**
**Line Size: 64 bytes**
**Block Size: 64 bytes**
**Read and Writes arrays: 4 x 4**
**Total number of bytes in a row/col/block = 64 bytes**

| | **Row-major access (UArray2)** | **Column-major access (UArray2)** | **Blocked Access (UArray2b)** |
|---|---|---|---|
| **90-degree rotation** | **3** | **3** | **1** |
| **180-degree rotation** | **1** | **6** | **1** |

**Row-major access(UArray2):**

- **90-degree rotation:** Rotating an image 90 degrees with row-major access means reading it row-wise -with a stride of 1- from the read array and writing it column-wise -with a stride of 4- into the write array. Based on the assumptions above, when the empty cache is asked for the data at index(0, 0) of the read array it misses and then reads the whole first row of the read array into a line in the cache since the size of the row equals the size of a block (line) in the cache and a row in an array has good spatial locality. It then tries to write that to index(0, 3) of the write array and misses but then reads the whole first row of the write array into a line in the cache. The next 3 subsequent attempts to read from the read array are hits since they exist in the cache but the next attempt to write to the write array misses because there is a stride of 4. This row-major read -> column-major write results in **20 misses** in a 4 x 4 array.
- **180-degree rotation:** Rotating an image 180 degrees with row-major access means reading it row wise -with a stride of 1- from the read array and writing it row wise (with a stride of 1) into the write array. Since this is a row-major read -> row-major write format and a single row of both the read and write arrays fits into the cache at the same time because they are contiguous in memory and enjoy good spatial locality there is a miss only when the cache tries to access data for the first time on a new row. This happens for both arrays and results in **8 misses**.

**Column-major access(UArray2):**

- **90-degree rotation:** Rotating an image 90 degrees with column-major access means reading it column wise (with a stride of 4) from the read array and writing it row wise (with a stride of 1) into the write array. This results in a column-major read -> row-major write which is similar to rotating an image 90 degrees with row-major access. Hence there are **20 misses** but majority of the misses happens when reading from the read array this time as to when majority of the misses occurred when writing with a row-major 90 degree rotation.
- **180-degree rotation:** Rotating an image 180 degrees with column-major access means reading it column wise (with a stride of 4) from the read array and writing it row wise (with a stride of 4) into the write array. Since both cases try to access memory that is not

necessary contiguous memory, they both suffer poor spatial locality. In general there will be a miss for every single read and write resulting in a total of **32 misses.**

**Blocked Access(UArray2b):**

- **N-degree rotation:** Rotating an image N degrees with blocked access means rotating each block individually before moving on to the next block. Since each block is contiguous in memory and fit in a line in the cache according to the assumptions made above, there will be a miss only when moving onto a new block. This happens both when reading and writing and hence will result in a total of **8 misses**. Also, since a block is always handled before moving on to a new block, the angle of rotation doesn't affect the number of misses because the needed block will always exist within the cache after the first miss. Hence both the 90 and 80 degree rotations will result in 8 misses each.

**RANKING:**

Based on the number of misses, methods with the same number of misses were given the same rank. Hence:

- Row-major access (UArray2) 90-degrees: 3
- Row-major access (UArray2) 180-degrees: 1
- Column-major access (UArray2) 90-degrees: 3
- Column-major access (UArray2) 180-degrees: 6
- Blocked Access (UArray2b) 90-degrees: 1
- Blocked Access (UArray2b) 180-degrees: 1