HW3: Locality and the Costs of Loads and Stores
Bill Yung and Dani Kupfer

Part D: Analyze Locality and Predict Performance

Expected cache hit rate for reads:

	row-major access	col-major access	blocked access
90-degree rotation	2	2	1
180-degree rotation	1	3	1

180-degree rotation with row-major access will have one of the highest hit rates. Within memory, the UArray2 is represented as a one-dimensional array and organized by row major order. When the cache reads in each element, the read-for-write stored is grouped by rows, which allows for good spatial locality for row major access. By transforming the image 180 degrees, the program simply takes one row and inserts it into another entire row of a new array. Therefore, the cache hit rate will be high both in terms of reading from the old array (using row-major order) and then writing into the new array (essentially using row-major order by default).

Blocked access works similarly, which is why it is also considered to have a high hit rate. In the same way that memory is stored by rows in a UArray2, the UArray2b blocks off data in order to achieve good spatial locality. Elements within the same block are close together in memory, as they are stored within a single UArray. When data is read, the cache will store the information by blocks. Therefore, reading through a block will have a high hit rate because the majority (if not all) of data already exists in the cache. Since the program will then write by block as well, this part of the program will also produce a high hit rate, which is the reason for its efficiency. There is no difference between 90-degree and 180-degree rotations (unlike row-major and col-major), because all data is stored and transferred by blocks, meaning that the different transformations do not impact the spatial locality.

As mentioned above, reading in via row-major access will have good locality; however, when the program does a 90 degree rotation, writing into the new array is essentially column-major access. Since the UArray2 is a one-dimensional array in memory and is grouped by rows, constantly accessing different columns will require the cache to continuously evict its data and have a low hit rate. Rotating an image by 90 degrees using column major is essentially a flipped version of a 90-degree rotation using row-major access. This time, reading in data by column major has many cache misses (since it continuously jumps from one row to another), but when writing in the data to the new array, it is done so by rows.

Rotating 180 degrees using column-major access has the lowest hit rate of all possible combinations. Reading and writing in both have many cache misses because they both require constant jumping between different rows.

Expected Work Per Pixel:

Kind of rotation	adds/subs	multiplies	divs/mods	compares	loads	hit rate	stores	hit rate
180 row-major	6	2	0	0	1	(blocksize-size_of_elt)/ blocksize	1	(blocksize-size_of_elt)/ blocksize
180 col-major	6	2	0	0	1	size_of_elt/ blocksize	1	size_of_elt/ blocksize
180 block-major	6	4	8	2	1	(blocksize-size_of_elt)/ blocksize	1	(blocksize-size_of_elt)/ blocksize
90 row-major	4	2	0	0	1	(blocksize-size_of_elt)/ blocksize	1	size_of_elt/ blocksize
90 col-major	4	2	0	0	1	size_of_elt/ blocksize	1	(blocksize-size_of_elt)/ blocksize
90 block-major	4	4	8	2	1	(blocksize-size_of_elt)/ blocksize	1	(blocksize-size_of_elt)/ blocksize

In terms of the hit rates for loads and stores, blocksize is representative of how much data can be stored in a single lined cache. For row-major and block-major, every element in the cache will generate a hit except for the first element. After the initial load, the rest of the elements within the row or block will exist in the cache and result in high hit rates. For column major, the program will constantly access data in different groups of memory, resulting in a low hit rate. While reading or writing in data, column major's next element is always on the next row, causing a cache miss and an additional load to get the next corresponding data element.

Expected Speed:

	row-major access	col-major access	blocked access
90-degree rotation	3	3	2
180-degree rotation	1	4	2