Cheng Li and Javi Gonzalez
COMP 40
HW 3 Implementation Plan: locality

Implementation plan for ppmtrans

Data structures:

- UArray2b will be represented as a 2D array whose elements are blocks, and each block is represented by a single Uarray. The index of elements in the Uarray will be coded so it corresponds to the index in UArray2b used for clients.

02/18

1. Set up files and make sure everything compiles correctly **(10 mins)**
    a. Code the implementation of uarray2b.h **(3 hours)**
        i. Test each method in the interface to make sure it works
        ii. Run a2block.c to make sure the uarray2b struct can be successfully applied on it.
    b. Code the implementation for a2plain.c **(2 hours)**

02/19

2. Come up with algorithms for the rotations and how are the mappings going to affect these
    a. Row major is going to have its own. **(3 hours)**
    b. Column major will have its own. **(1 hour - it'll be similar to row major)**
    c. Block major will have its own. Remember that for block major first we need to move the given blocks to its landing position and then we rotate the data stored in the block accordingly (Note: the rotation in low level was done by manipulating the index of elements in an 1D array). **(3 hours)**

02/20

3. Use **Pnm_ppm Pnm_ppmread** to read the given image
    a. Store this image in a 2D array that will represent the original image. **(1 hour)**
    b. Map the image with the given rotation and assign each cell in the array to a new position in the new "rotated 2D array". **(5 hours)**
        i. To perform the rotations we will create a new 2 dimensional array and we will use the **A2_methods apply function** and pass in the new destination array as the A2 array2 and we will pass in the current looping index in the original image's 2D array as the **A2Methods_object *ptr,** so that we can use the At function and access the original image at the given position.
        ii. We will initially account for the 0, 90 and 180 degree rotations (if we have time we will account for the rest - flips, transposes…etc).

1. All of these rotations will have different algorithms (we wont use the 180 twice for 0 degree rotation, nor the 90 for the 180, 270 or 0.

c. Write this 2D array in the **binary ppm** format using **void Pnm_ppmwrite** and direct it to an output file. **(1 hour)**
d. Make sure to **free** the memory allocated with this interface also with it. **(15 mins)**

02/21 && 02/22

4. Account for the rest of extra credit rotations **(3-4 hours)**
5. Test thoroughly and Debug our program **(1.5 days)**

Testing

1. After accomplishing 1.a and 2.a. We generate a random ppm, and store this image in the form of UArray2b_T struct. We then print them out to see if the struct is working.
2. Read in an empty file
3. Read in a image that represents a square (should not be affected by rotations)
4. Read in a file that contains a vertical line and rotate it
5. Play with some symmetric and asymmetric images to see how they should behave (first perform the rotations ourselves and then compare them against the code)
6. Read in a small image but stored in a 64KB block array. Perform rotations. Use valgrind to check the memory when it comes to the margins.
7. Read in larger images offered by the cs40 library
8. Run all of these tests with valgrind
9. Run the program using the timer to make sure that the rotations work as they should

Part D (experimental) - Analyze locality and predict performance

For this part of the assignment we have to estimate the expected cache hit rate for reads, while assuming that images do not fit entirely on the cache. Given that we only have to estimate the rate of hits for the reads, and not the writes, the degree of the rotation won't affect this rate (given that when we are rotating the image we are only writing).

Mapping functions overall:

In row-major mapping the information is looped through as like a one dimensional array (all of the element data are contiguous and we can read everything "in order). In this case we could just store part of the array in the cache at once, and the cache would miss the first one but hit the rest of the reads. For this reason row major mapping would be perfect with spatial location.

Column major mapping

The number of read misses in the column major mapping is the worst out of all of the types of mappings. This is because data is not being read contiguously as in memory. One important

factor to take into consideration is that the longer the rows are, the further apart the columns will be, and the worst case is when the first element of each column is very far apart from each other so that what was read into the cache will never be found again until it has to be evicted, which means no cache hit will happen. This is significantly worse than the other two mapping functions as they both have memory stored contiguously, resulting in a more efficient cache hit rate.

Block major mapping

It allows us to work with smaller blocks of information at once, hence we can just store everything in the block in the cache at once. This means that we could have subsequent reads throughout the block (1 initial miss but the rest of the block all hits), in the prerequisite that a block must be smaller than a cache line. If a block takes no entire memory in a cache line, we SUPPOSE the cache line will continue to read contagious data until it was 'filled up' (if cache line will just stop reading after single block, the performance of block major mapping will be worse than row major but better than column major). Essentially, the UArray2b is represented by a UArray2 composed of UArray as blocks, and the UArray2 is represented by multiple UArrays wrapped up by a Uarray (the leading rows). It means that UArray2b, in memory, is multiple UArrays connected to each other head after tail. This would result in perfect spatial location and as good performance of the cache as row major.

RANKINGS	Row major	Column major	Block major
90 degree rotation	1	2	1
180 degree rotation	1	2	1

When the cache line stops to read in contagious data after an entire block was read:

RANKINGS	Row major	Column major	Block major
90 degree rotation	1	3	2
180 degree rotation	1	3	2