Jack Burton jburto05 & Alekha Rao arao12
2/12/24

Locality Design Document

Ppmtrans Implementation Plan

1. *Make test file with main function and mess around with Pnm.h to get familiar with it*
2. Modify Makefile to compile ppmtrans (and try to compile given ppmtrans, ensuring no linker errors/only errors from parts that haven't been written in the code yet)
3. Complete argument handling of flags portion of ppmtrans
    a. Add specific error for 270-degree rotation *(maybe removed later)*
    b. Add to last "else" to make rotation of 0 degrees the default option

Testing Plan:

- Print statements in each argument flag check, and ensure that correct if statements are reached for each flag
    - Try this with 1, 2, and 3 argument flags (of the various types)

4. Handle file argument portion of ppmtrans
    a. Since i is defined outside of for loop, open file at argv[i]
        i. Assert that filePointer is non-NULL after opening for reading
    b. Add fclose() at end of main loop to ensure no memory leaks

Testing Plan:

- Try various combinations of 1, 2, and 3 argument flags plus a filename, ensuring that the file is opened in all of these cases
- Try argument flag combos plus an nonexistent file, ensuring that assert catches this and throws exception
- Try argument flag combos with an empty file, ensuring that assert throws an exception.
- Try valgrind with valid files and ensure all memory is freed

5. Handle rotations with a rotation function that takes int degrees, file pointer as arguments
    a. Create empty rotation function and assert each variable
    b. First, implement function for rotation of 0 degrees
        i. just pnm_ppmread and pnm_ppmwrite the same file pointer to standard output
    c. Then add if statements to check if rotation is 90 or 180, and set "math" variables accordingly from spec geometry section
    d. Then pnm_ppmread image into specified UArray/UArray2b, perform transformation of pixels using specified/default map function and "math" variable, and pnm_ppmwrite to stdout

   e. Pnm_ppmfree at end of rotation function, AND free created UArray/UArray2b (either directly from function or with memory freeing helper)

Testing Plan:

- Pass NULL as arguments and ensure exceptions are thrown
- Make small (10x10) images to test on 0/90/180 degree rotation to ensure the rotations are working. Debug accordingly.
- Valgrind on small images, ensure no memory leaks
- Test given CS40 images on the 0 degree rotation, diff/use display to ensure the same image is output
- Valgrind with 0-degree rotation, ensure no memory leaks
- Test given CS40 images on 90/180 degree rotations, use display to ensure image is properly rotated 90/180 degrees, respectively
- Valgrind with 90 and 180 degree rotations ensuring no memory leaks

6. Review page about Timing Instructions and mess with given timing test file to get familiar with implementation
7. Include cputiming.h in ppmtrans and ensure compilation succeeds using makefile (add cputiming.o to makefile rule for ppmtrans)
8. Make time function
   a. Create timing function which takes filename and writes double to an output file named “filename” (and fclose)
   b. Go to rotation functions and add condition to check if time flag was set
      i. If set, add code to Start CPUTimer just before rotation and stop it just after (and pass that output double to timing function)
      ii. Free CPUTimer after recording value

Testing Plan:

- After just creating time function, execute it from main() and see if it will correctly create/write to client-specified output file in argos
- Valgrind to ensure no memory leaks
- Put CPUTimer code in rotation functions and printf the double with the time to ensure CPUTimer is working correctly
- Valgrind to make sure CPUTimer doesn’t leak memory
- Pass CPUTimer double as input to time function and test with 10x10 and large CS40 given test images, ensuring correct time values are written to specified output file

9. Create rotate 270, flip vertical, flip horizontal, and transpose functions, if extra time after testing everything else thoroughly

Jack Burton jburto05 & Alekha Rao arao12
2/12/24

Part D Estimates/Expectations

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

Justifications

90-degree vs 180-degree:
Since we are only considering the expected cache hit rates for reads, they shouldn't differ too much between 90 and 180-degree rotations. Instead, the hit rates are affected much more by the type of mapping that is used.

Row-major access:
Our implementation of UArray2 uses a "parent" UArray with each element of that parent being a "child" UArray. Therefore when doing row-major mapping, you are accessing memory that is not contiguous, since you're effectively accessing different "child" UArrays to get each element in a row. This will mean having to access the memory directly more often since no elements are actually contiguous, which will create a much higher (almost 100%) miss rate. This will mean almost no locality.

Column-major access:
For column-major access, with our structure for UArray2s you will remain in a single "child" UArray for the entire length of it. Therefore when doing column-major access, a child UArray could be pushed to the cache and accessed from there repeatedly until you go through all of its elements, and *then* switched for another child UArray from memory. This will create a lower miss rate than row-major access. Column major access will have better locality than row-major, but still not as good as blocked.

Blocked access:
Blocked access using UArray2b's should have the highest hit rate (lowest miss rate) because the image's pixels will be divided into blocks, presumably of a width > 1, and each block will be put into the cache while it is mapped through, and *then* replaced with another block after that. Overall in the image, there should be less blocks than columns (going back to our presumption that the blocks have a width > 1 pixel), so this will ultimately mean even less accessing of direct memory to change in the cache versus column-major access. The blocked access will have the highest locality of any mapping method. Therefore, block major will have the best locality compared to both row and column major.