CSCI 210 Assn. 5: Caching effects

Lab note: *You may work with up to one other person on this assignment and submit your solution together, or you may work alone. The write-up is due at 5:00pm on Friday, April 30. It is worth 40 points.*

Objectives

- to see how caching can affect the efficiency of a computer program.
- to learn how a program can be modified to take advantage of the cache more fully.

A Observations

You are working on the development team of an image-processing program (such as GIMP or Photoshop), and your supervisor has requested a report investigating whether the component for rotating the image can be improved.

The "`getcs 210 5b`" command will fetch an image-processing program for you to use. You should be able to compile and run these files.

When you select "Open..." from the File menu, the program will start in a directory holding several copies of an image of varying sizes. (It can load other GIFs and JPEGs, too.) After an image is opened, you can select "Rotate 1" from the Image menu (or press the '1' key) to rotate the image clockwise. Note that when you rotate the image, the program shows you how much time the rotation took in the label just below the menu bar. You'll investigate the performance of the rotation code in this assignment.

Rotating an image is a simple matrix operation, taking an $n \times n$ matrix A (each entry being the color of a pixel) and rotating each element $A_{i,j}$ to $A_{j,n-1-i}$. Here's an example:

$$\text{rotate} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{pmatrix}$$

You can see, for example, that $A_{0,0}$ has rotated into $A_{0,2} = A_{0,3-1-0}$.

The `ImageTransform.java` file contains the code for rotation; this is the code you are investigating. Because of the Java API's method of handling images, the Java code actually stores the image as a one-dimensional array, where entry $A_{i,j}$ is stored at array index $in + j$.

Make a table of how much time it takes to rotate each image size. When you measure time, you should *not* simply take the first reading you get, because the first reading could be influenced by virtual memory and cache, where subsequent rotations would not be. A good approach is to rotate the image several times quickly (more than 5), then computing the average time over the last three times you rotated it.

n	time	$\frac{\text{time}}{n^2}$
200		
400		
600		
800		
1200		
1600		
2000		

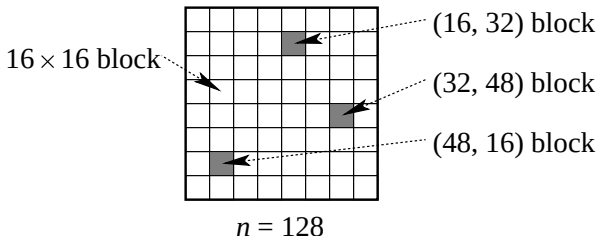
Since the image rotation process is an $O(n^2)$ algorithm, you would expect the time divided by n^2 to be roughly the same. Note, however, that it is not constant; this is a result of how poorly the rotation code works with the cache.

B Modifications

Your job for the second part of the lab is to modify the rotation code to avoid this cache effect while still maintaining the correctness of the code.

To help you with this, let me give you a solution to a similar problem where caching effects are noticeable. In this problem, our goal is to find the combination of i, j, and k so that the sum $A_{i,j} + A_{j,k} + A_{k,i}$ is minimum.[1] The naïve approach to this is in Figure A5.1(a). Because this code uses an $O(n^3)$ algorithm, we would expect the time taken by this code divided by n^3 to be roughly constant as n increases. But measurements would demonstrate an effect similar that seen in Part A: The time divided by n^3 is not constant, due to caching effects.

To address this issue, we break the matrix into 16×16 blocks, each easily small enough to be managed by the cache.



Then we analyze each group of three blocks individually. Since all three blocks can fit into the cache with no problems, the caching effect is avoided. In Figure A5.1(b), `bi`, `bj`, and `bk` iterate through the blocks, while the inner loops (over `i`, `j`, and `k`) iterate through indices within the blocks.

Your solution should be able to work quickly for any image size, not simply where n is a multiple of 16. Note that `ImageTransform.java` provides three methods for writing rotation code, selected through the menu. This allows you to compare the speed of various algorithms during a single run of the program.

Your write-up for this assignment should be the report you would submit to your supervisor. Your report should present the two algorithms implemented (the original one tested and your modified version), the results of testing both algorithms, discussion of these results quantifying how the improvement helped (and possibly questions raised by the work that you still do not understand), and a recommendation of which to use and why.

[1] If you want an application motivating this problem, consider the matrix to be a table of distances between locations. We're looking for the set of three places that are closest together.

```
int minTriadNaive(int[] a) {
    int min = Double.MAX_VALUE;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            for(int k = 0; k < n; k++) {
                int entry = a[n * i + j] + a[n * j + k] + a[n * k + i];
                if(entry < min) min = entry;
            }
        }
    }
    return min;
}
```

(a) The naïve approach

```
int minTriadGood(int[] a) {
    final int BLOCK = 16;

    // if n is below the block size, handle matrix normally.
    if(n <= BLOCK) return minTriadNaive(a, n);

    int min = Double.MAX_VALUE;
    // iterate through all sets of three blocks
    for(int bi = 0; bi < n; bi += BLOCK) {
        for(int bj = 0; bj < n; bj += BLOCK) {
            for(int bk = 0; bk < n; bk += BLOCK) {
                // iterate through the indices of the three selected blocks
                for(int i = bi; i < bi + BLOCK; i++) {
                    for(int j = bj; j < bj + BLOCK; j++) {
                        for(int k = bk; k < bk + BLOCK; k++) {
                            int entry = a[n * i + j] + a[n * j + k] + a[n * k + i];
                            if(entry < min) min = entry;
                        }
                    }
                }
            }
        }
    }
    return min;
}
```

(b) The improved approach

Figure A5.1: Two solutions to finding the closest three points in a matrix.