Hw2

202111161 전사빈

Implementation results

Img_128.bmp	Img_256.bmp	Img_512.bmp	Img_768.bmp	Img_1024.bmp
4.42	4.66	4.63	4.66	5.24

The graph below shows speedup between baseline and optimized convolution code.

The largest image showed best performance.

Optimization approach

I optimized the code using 6-7 ways.

1. Reduce procedure call 1.22/1.0
   First, I approached to reduce procedure calls. In given code, the function 'convolution' had been called for each loop. That is, the procedure had been called for weight*height time. I thought it was inefficient factors that make the execution time slower. So, I deleted the convolution function, and inserted that procedure in filter_optimized function. I think this approach raised the performance through not calling unnecessary procedures. the speedup with that approach showed 1.22/1.0.

2. Convert variable type 1.28/1.22
   In the given code, r, g, b was referenced as double type. The intensity of pixel has integer value. So, expression of floating point was unnecessary. But converting to integer type has changed the intended result. The result need to express floating point. So, I convert the type to float, which has 4 byte value while double has 8 byte value. I thought this approach would reduce access time of data. But it did not show so much performance increase. The speedup through this approach was 1.28/1.22.

3. Switch array approach order 1.42/1.28
   In given code. The function approached to array with stride width. It does not cache friendly code, decreasing spatial locality. So, I switch the order of array approach to stride 1, switching the loop counter x and y. with this approach, the code utilize the spatial locality, decreasing potential cache miss. The speedup with this approach was 1.42/1.28.

4. Reduce 'if' 1.44/1.42
   At the last of convolution, there are six conditional command to limit the intensity value of filtered pixel. Conditional commands make unnecessary delay. So, it was necessary to minimize those commands. I used conditional operator '?'. With conditional operator, I can make the code shorter. But it does not make meaningful result. It just shortened the code, not making faster the execution.

5. Loop unrolling/reduce strength 2.86/1.44
   Loop unrolling was one of the most critical ways in optimizing. In the given code, there are width*height*9 of loop. The loop can decompose into height*width outer loop and 3*3 inner loop. I thought 3*3 inner loop is always executed while outer loop is not always executed same time due to image size. So, I unrolled the inner 3*3 loop. And with application, I revised some arithmetic operations that duplicated many times. I referred that duplicated arithmetic operations on the top of the code as a variable. These approaches showed almost twice of speedup. 2.86/1.44

6. Register variable 5.10/2.86
   In the code, there are some variables that called many times. I thought the execution would faster If those variables are stored in the register when it is declared, And I found out that c supports the register variable type. Rows and columns of the array, height and width of the image, R, G, B intensities of each pixel were declared as register variables to reduce the latency when the variables were called. The result showed a performance increase of about twice as much and showed more effect than expected.