Nicholas Carnevale
CSE 201
Programming Project 3

After initially testing the given functions I found that on average, the gamma function and the blur function were the two that were taking over >20 percent of the total time consistently.

Looking at the two functions, I decided to try reducing the time of the gamma function first, because it looked less complex. These are the rest of the techniques I tried out during this project. ALL OF MY OUTPUT IS COMMENTED IN MY prog3.cpp.

Optimization Techniques Used in PP:

1. Reduce function calls to Gamma Function.

In this technique I used the commented out pixel, height, and width variables to reduce the function calls to in_rows variable.
The result was an improvement of about 2000 cycles on each image.

2. Apply 2x1 loop unrolling to Gamma function.

In this technique on each loop I applied the gamma function to two pixels at once. I also carried over the reduced function call optimization.
This resulted in about a 5000 cycle improvement on top of the reduced function call improvement.

3. Apply 4x1 loop unrolling to Gamma function.

In this technique at each loop iteration we applied the gamma function to 4 pixels.
The result was about the same as the 2x1 loop unrolling for the first image, but there was about a 500-1000 cycle improvement for the rest of the images.

4. Apply 2x1 loop unrolling to the blur function

In this technique at each loop iteration we blurred two functions.
This technique only resulted in about a 1000 cycle improvement from the base function.

5. I then tried a 2x1 loop unrolling with an inner loop to process the 2 pixels in the blur function.

The result was about another 1000-2000 improvement on top of the improvement from before so I was happy about that.

6. Apply the 2x1 loop unrolling to the Tint function

This resulted in a 5000 cycle improvement on an already fast function which was very impressive.

The average total time was around 58,000 cycles after the optimizations compared to the 75,000 cycle time without any optimization.

The cache performance (miss rate) of the functions applyGamma, applyTint, and applyBlur if the cache is direct-mapped, the size of the cache line is 64 bytes, and the size of the cache is 32KB.

Direct mapping means E = 1
B = 64
Size of cache = 32 KB
S = 500 sets

Each pixel takes up 16 bytes because it has an array of 4 ints which are the colors.

For the apply Gamma function:

In_rows[0][0] will be a cold miss. -> Will load In_rows[0][0] - In_rows[0][3] into block 1
In_rows[0][1] will be a hit.
In_rows[0][2] will be a hit.
In_rows[0][3] will be a hit.
In_rows[0][4] will be a cold miss. -> Will load In_rows[0][4] - In_rows[0][7] into block 2

And so on with a miss rate of 25%.

For the apply tint function.
The miss rate will be the same because the same Pixels are being accessed at each iteration.
25% miss rate.

The given apply blur is very complex and will almost definitely have a higher miss rate.
I will calculate the miss rate for the middle rows and middle columns because that is the majority of the calls compared to the first row and the last row only being two rows of the total. (the function is defined differently for those rows but essentially the same data is called.

At each iteration:
in_rows[i-1][j-1].bgra[0] + in_rows[i-1][j].bgra[0] + in_rows[i-1][j+1].bgra[0] + in_rows[i][j-1].bgra[0] + in_rows[i][j].bgra[0] + in_rows[i][j+1].bgra[0] +
in_rows[i+1][j-1].bgra[0] + in_rows[i+1][j].bgra[0] + in_rows[i+1][j+1].bgra[0].
The max image width is > 500 therefore two points at the same column level cannot be in the cache at the same time. Therefore at each iteration, everything with in_rows[i-1][x] or in_rows[i+1][x] will always miss. I think because the last call is in_rows[i+1][j+1].bgra[0]. And the next iteration it is in_rows[i-1][j-1].bgra[0], they will both result in the cache being loaded differently and a miss. I think this function may have a 100% miss rate but I am not exactly sure. At very least the miss rate is 66.6%.