CENG444 PROGRAMMING ASSIGNMENT #3

In this homework, it is expected to blurring images with given Gaussian blur kernel matrix. I designed my algorithm based on PNG images. In PNG images, there is a header in hexadecimal format that specifies features of the PNG images. Firstly, I tried to <png.h> library to extract different channels of a PNG however, I cannot work with it as I desire. By using <png.h> library, the user has to manipulate each decoded piece of the PNG image such as manipulating transparency, color depth. Also, with respect to the image such as RGB or gray-scale image, user has to take different actions. Instead, I found another library called as stb [1], each image format can be serialized with respect to its channel amount. In addition, user can implement their image controlling code without depending on the image format. For example, I can use control each image format as if each of the PNG images has 3 channels as RGB. The other channels have already been embedded into these RGB channels.

We are free to apply any policy choice for boundary pixels of the image. Thus, I keep them as it is without manipulating them. Also, It is said that we should not change the original image. Thus, I have created 3 different arrays to hold each of the 3 channels.

To compile the given code, user has to write '**gcc 283078027.c -o 283078027 –lm –fopenmp**' from the command terminal. It will enable the required flags with –lm and –fopenmp specifications. Then, '**283078027**' executable file will exist in the same directory, and user can run this with '**./283078027**'.

To optimize the execution time of the code, I have tried different configurations both taking care of the different scheduling types such as dynamic scheduling with 50 and 100 chunks, guided and static scheduling and being aware of the algorithm workflow such as mappings, scans and fusing. The shared code is the optimized version that I can reach eventually. There is an important point on this optimization problem. Since PNG images with 8-bit pixel values (after tuning even if bit_depth = 1,2,4, or 16) have the image size which is not as big as the previous homework's workloads, execution time is smaller compared to the previous assignments. Thus, the overhead for each thread is an important factor in terms of execution time. For example, even if guided or dynamic scheduling of for the threads is beneficial with respect to the static scheduling, static scheduling results in best execution time performance due to its simpler structure and lower overhead.

First of all, I have copied all 3-pixel channels into 3 different 2D arrays. Afterwards, instead of rewriting the calculated results to the 3 different 2D arrays, I wrote the blurred pixel results into resultant arrays. I have decreased 1 step, and it has increased the performance little bit. In addition, I have arranged 3 different 2-nested for loops and calculated the blurring for each of the arrays separately. I have done this to prevent the load of other arrays and utilize the spatial locality of the caches fully. I did not separate the arrays for creation of the 3 2D arrays corresponding to channels since it did not provide any increase on the performance. Also, once the original image array is loaded into cache, writing each pixel of the original imagine to the 3 different 2D arrays is easier instead of fetching it from the DRAM for 3 different loops. The execution time of the algorithm for different number of threads is shared in the figure 1.

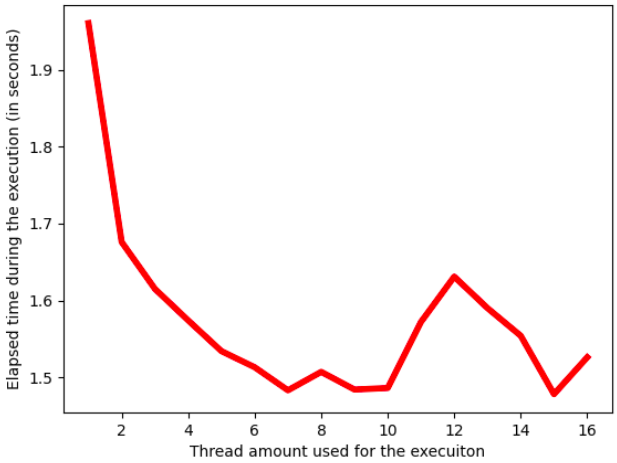


Figure 1: Execution time of the algorithm with different number of threads (for the forrest.png shared in the 283078027.zip file)

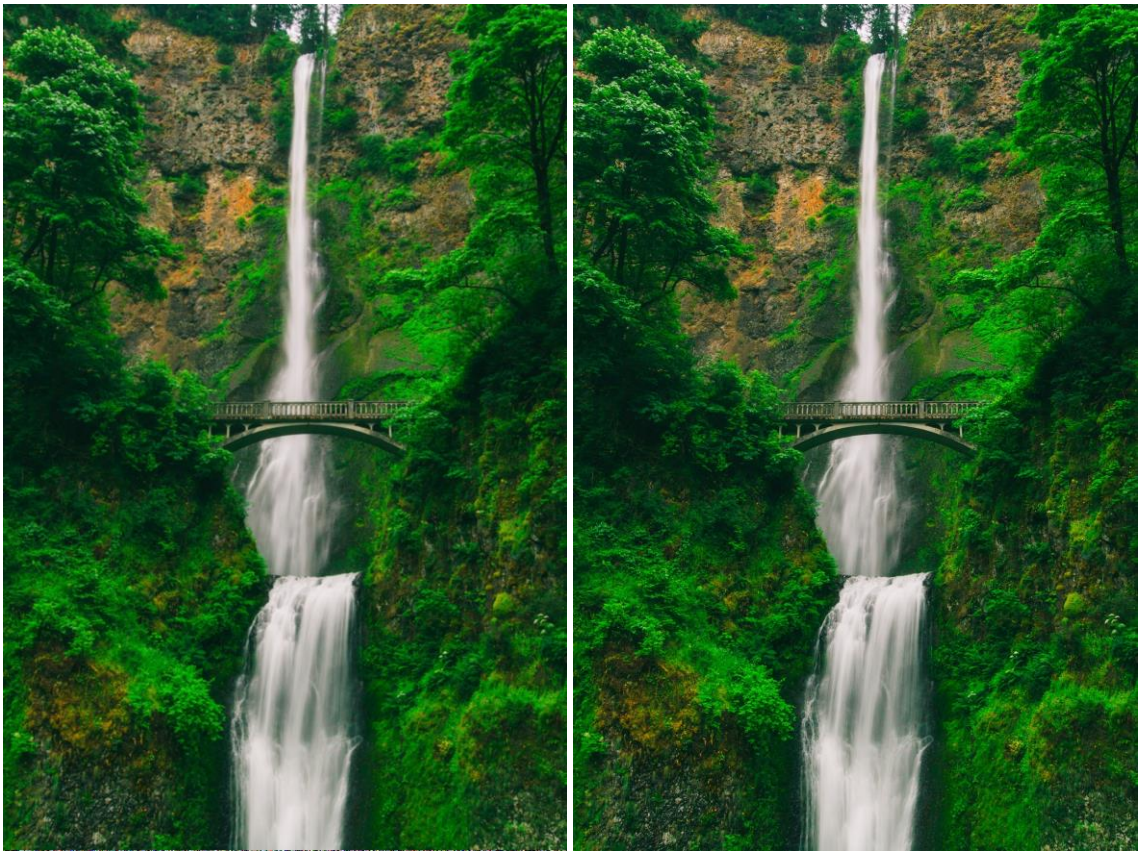
As it is expected for the first 7 configuration, execution time is decreased. Different from the previous programming assignments, I cannot observe too much performance optimization with enabling different cores to execute in parallel. The main reason behind this is that multi programming overheads such as synchronization and overheads of the threads limits the performance even if we increase the number of cores for computation. However, if this blurring process is done for multiple PNG image, the execution time gap between different number of threads will increase. Furthermore, the performance increase highly depends on the pixel amount of the PNG image. If the resolution of the image is huge meaning that both more bit-depth and pixel amount, the computation workload will increase and parallelization with dividing computation workload and assigning threads to different cores will be more useful.

As I mentioned, I designed my code to increase cache utilization and removing unnecessary workloads by dividing computation missions into threads, mapping these threads to the cores and fusing the total process as much as possible. In my computer, there are 12 cores and 3-leveled cache configuration in my computer (while 1st level cache is separate as instruction and data, 2nd and 3rd level caches are unified). Also, I want to emphasize that while increasing amount of pixel the observable effect of the blurring will decrease. I have shared different PNG images with its original and blurred versions in the appendix part in addition to screenshots of my computer hardware and performance results.

[1] : https://github.com/nothings/stb/blob/master/stb_image.h

APPENDIX

Blurring illustrations where left one is blurred and right one is the original image.



Computer Hardware Specifications

```
burak@burak-laptop:~/Documents/ceng444/hw3$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   48 bits physical, 48 bits virtual
CPU(s):                          12
On-line CPU(s) list:             0-11
Thread(s) per core:              2
Core(s) per socket:              6
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       AuthenticAMD
CPU family:                      23
Model:                           96
Model name:                      AMD Ryzen 5 4600H with Radeon Graphics
Stepping:                        1
Frequency boost:                 enabled
CPU MHz:                         1330.902
CPU max MHz:                     3000,0000
CPU min MHz:                     1400,0000
BogoMIPS:                        5988.76
Virtualization:                  AMD-V
L1d cache:                       192 KiB
L1i cache:                       192 KiB
L2 cache:                        3 MiB
L3 cache:                        8 MiB
NUMA node0 CPU(s):               0-11
```

Performance results shown in figure 1 in addition to compile and run commands.

```
burak@burak-laptop:~/Documents/ceng444/hw3$ gcc 283078027.c -o main -lm -fopenmp
burak@burak-laptop:~/Documents/ceng444/hw3$ ./main
The elapsed time to run with 1 amount of thread is 1.961 seconds
The elapsed time to run with 2 amount of thread is 1.676 seconds
The elapsed time to run with 3 amount of thread is 1.615 seconds
The elapsed time to run with 4 amount of thread is 1.574 seconds
The elapsed time to run with 5 amount of thread is 1.534 seconds
The elapsed time to run with 6 amount of thread is 1.513 seconds
The elapsed time to run with 7 amount of thread is 1.483 seconds
The elapsed time to run with 8 amount of thread is 1.507 seconds
The elapsed time to run with 9 amount of thread is 1.484 seconds
The elapsed time to run with 10 amount of thread is 1.486 seconds
The elapsed time to run with 11 amount of thread is 1.572 seconds
The elapsed time to run with 12 amount of thread is 1.631 seconds
The elapsed time to run with 13 amount of thread is 1.590 seconds
The elapsed time to run with 14 amount of thread is 1.554 seconds
The elapsed time to run with 15 amount of thread is 1.478 seconds
The elapsed time to run with 16 amount of thread is 1.526 seconds
```